Security in MySQL
Abstract

This is the MySQL Security Guide extract from the MySQL 8.0 Reference Manual.

For legal information, see the Legal Notices.

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

Document generated on: 2020-04-08 (revision: 65636)
# Table of Contents

Preface and Legal Notices ........................................................................................................................... v  
1 Security .......................................................................................................................................................... 1  
2 General Security Issues ................................................................................................................................. 3  
   2.1 Security Guidelines .................................................................................................................................. 3  
   2.2 Keeping Passwords Secure ....................................................................................................................... 4  
      2.2.1 End-User Guidelines for Password Security ....................................................................................... 4  
      2.2.2 Administrator Guidelines for Password Security .............................................................................. 6  
      2.2.3 Passwords and Logging .................................................................................................................. 6  
   2.3 Making MySQL Secure Against Attackers ............................................................................................... 7  
   2.4 Security-Related mysqld Options and Variables .................................................................................... 9  
   2.5 How to Run MySQL as a Normal User ................................................................................................... 9  
   2.6 Security Considerations for LOAD DATA LOCAL .............................................................................. 10  
   2.7 Client Programming Security Guidelines ............................................................................................. 13  
3 Postinstallation Setup and Testing ................................................................................................................... 15  
   3.1 Initializing the Data Directory .................................................................................................................. 15  
   3.2 Starting the Server ................................................................................................................................... 20  
      3.2.1 Troubleshooting Problems Starting the MySQL Server ................................................................... 21  
      3.3 Testing the Server .................................................................................................................................. 23  
   3.4 Securing the Initial MySQL Account ...................................................................................................... 25  
   3.5 Starting and Stopping MySQL Automatically .......................................................................................... 27  
4 Access Control and Account Management ....................................................................................................... 29  
   4.1 Account User Names and Passwords ......................................................................................................... 30  
   4.2 Privileges Provided by MySQL ................................................................................................................ 32  
   4.3 Grant Tables ............................................................................................................................................ 48  
   4.4 Specifying Account Names ...................................................................................................................... 56  
   4.5 Specifying Role Names ............................................................................................................................ 58  
   4.6 Access Control, Stage 1: Connection Verification .................................................................................. 59  
   4.7 Access Control, Stage 2: Request Verification ......................................................................................... 62  
   4.8 Adding Accounts, Assigning Privileges, and Dropping Accounts ......................................................... 63  
   4.9 Reserved Accounts .................................................................................................................................. 66  
   4.10 Using Roles ............................................................................................................................................. 67  
   4.11 Account Categories .................................................................................................................................. 74  
   4.12 Privilege Restriction Using Partial Revoke ............................................................................................ 77  
   4.13 When Privilege Changes Take Effect ..................................................................................................... 83  
   4.14 Assigning Account Passwords ................................................................................................................ 84  
   4.15 Password Management ............................................................................................................................ 85  
   4.16 Server Handling of Expired Passwords ................................................................................................. 96  
   4.17 Pluggable Authentication ....................................................................................................................... 97  
   4.18 Proxy Users ............................................................................................................................................ 102  
   4.19 Account Locking ..................................................................................................................................... 110  
   4.20 Setting Account Resource Limits ......................................................................................................... 110  
   4.21 Troubleshooting Problems Connecting to MySQL .................................................................................. 112  
   4.22 SQL-Based Account Activity Auditing ................................................................................................... 116  
5 Using Encrypted Connections .......................................................................................................................... 119  
   5.1 Configuring MySQL to Use Encrypted Connections ............................................................................ 120  
   5.2 Encrypted Connection TLS Protocols and Ciphers ............................................................................. 126  
   5.3 Creating SSL and RSA Certificates and Keys ........................................................................................ 132  
      5.3.1 Creating SSL and RSA Certificates and Keys using MySQL ............................................................ 132  
      5.3.2 Creating SSL Certificates and Keys Using openssl ......................................................................... 135  
      5.3.3 Creating RSA Keys Using openssl ..................................................................................................... 140  
   5.4 Connecting to MySQL Remotely from Windows with SSH .................................................................. 140  
6 Security Components and Plugins ................................................................................................................... 141  
   6.1 Authentication Plugins ............................................................................................................................. 142  
      6.1.1 Native Pluggable Authentication ....................................................................................................... 142  
      6.1.2 Caching SHA-2 Pluggable Authentication ....................................................................................... 143
6.8 MySQL Enterprise Data Masking and De-Identification ........................................... 352
6.7 MySQL Enterprise Firewall .................................................................................. 340
6.6 The Audit Message Component ........................................................................ 337
6.5 MySQL Enterprise Audit .................................................................................... 271
6.4 The MySQL Keyring ........................................................................................... 228
6.3 The Password Validation Component .................................................................. 216
6.2 The Connection-Control Plugins ....................................................................... 210
6.1 Security in MySQL.............................................................................................. 148
  6.1.3 SHA-256 Pluggable Authentication ................................................................. 148
  6.1.4 Client-Side Cleartext Pluggable Authentication ............................................. 152
  6.1.5 PAM Pluggable Authentication ........................................................................ 153
  6.1.6 Windows Pluggable Authentication ................................................................ 163
  6.1.7 LDAP Pluggable Authentication ...................................................................... 168
  6.1.8 No-Login Pluggable Authentication ................................................................ 186
  6.1.9 Socket Peer-Credential Pluggable Authentication .......................................... 189
  6.1.10 Test Pluggable Authentication .................................................................... 191
  6.1.11 Pluggable Authentication System Variables ................................................. 193
  6.2 The Connection-Control Plugins ....................................................................... 210
  6.2.1 Connection-Control Plugin Installation ......................................................... 210
  6.2.2 Connection-Control System and Status Variables ......................................... 214
  6.3 The Password Validation Component .................................................................. 216
  6.3.1 Password Validation Component Installation and Uninstallation ................... 218
  6.3.2 Password Validation Options and Variables .................................................. 218
  6.3.3 Transitioning to the Password Validation Component .................................... 227
  6.4 The MySQL Keyring ........................................................................................... 228
  6.4.1 Keyring Plugin Installation ............................................................................. 229
  6.4.2 Using the keyring_file File-Based Plugin ...................................................... 230
  6.4.3 Using the keyring_encrypted_file Keyring Plugin ......................................... 231
  6.4.4 Using the keyring_okv KMIP Plugin ............................................................... 232
  6.4.5 Using the keyring_aws Amazon Web Services Keyring Plugin ...................... 237
  6.4.6 Using the HashiCorp Vault Keyring Plugin .................................................... 240
  6.4.7 Migrating Keys Between Keyring Keystores .................................................... 247
  6.4.8 Supported Keyring Key Types and Lengths ..................................................... 249
  6.4.9 General-Purpose Keyring Key-Management Functions ................................. 251
  6.4.10 Plugin-Specific Keyring Key-Management Functions ................................... 258
  6.4.11 Keyring Command Options ....................................................................... 259
  6.4.12 Keyring System Variables ........................................................................... 261
  6.5 MySQL Enterprise Audit .................................................................................... 271
  6.5.1 Audit Log Components ................................................................................... 272
  6.5.2 Installing or Uninstalling MySQL Enterprise Audit ........................................ 272
  6.5.3 MySQL Enterprise Audit Security Considerations ........................................ 274
  6.5.4 Audit Log File Formats .................................................................................. 274
  6.5.5 Audit Log Logging Configuration .................................................................... 293
  6.5.6 Audit Log Filtering .......................................................................................... 302
  6.5.7 Writing Audit Log Filter Definitions ................................................................ 305
  6.5.8 Legacy Mode Audit Log Filtering .................................................................. 317
  6.5.9 Audit Log Reference ...................................................................................... 319
  6.5.10 Audit Log Restrictions .................................................................................. 337
  6.6 The Audit Message Component ......................................................................... 337
  6.7 MySQL Enterprise Firewall ................................................................................ 340
  6.7.1 MySQL Enterprise Firewall Components ....................................................... 341
  6.7.2 Installing or Uninstalling MySQL Enterprise Firewall .................................... 341
  6.7.3 Using MySQL Enterprise Firewall ................................................................. 344
  6.7.4 MySQL Enterprise Firewall Reference .......................................................... 348
  6.8 MySQL Enterprise Data Masking and De-Identification .................................... 352
  6.8.1 MySQL Enterprise Data Masking and De-Identification Components ............ 354
  6.8.2 Installing or Uninstalling MySQL Enterprise Data Masking and De-Identification 354
  6.8.3 Using MySQL Enterprise Data Masking and De-Identification ........................... 355
  6.8.4 MySQL Enterprise Data Masking and De-Identification User-Defined Function Reference ................................................................. 360
A MySQL 8.0 FAQ: Security ......................................................................................... 371
Preface and Legal Notices

This is the MySQL Security Guide extract from the MySQL 8.0 Reference Manual.

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

**Legal Notices**

Copyright © 1997, 2020, Oracle and/or its affiliates. All rights reserved.

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

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

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

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

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

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

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

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

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

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

Documentation Accessibility

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

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit
https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab.
Chapter 1 Security

When thinking about security within a MySQL installation, you should consider a wide range of possible topics and how they affect the security of your MySQL server and related applications:

- General factors that affect security. These include choosing good passwords, not granting unnecessary privileges to users, ensuring application security by preventing SQL injections and data corruption, and others. See Chapter 2, General Security Issues.

- Security of the installation itself. The data files, log files, and all the application files of your installation should be protected to ensure that they are not readable or writable by unauthorized parties. For more information, see Chapter 3, Postinstallation Setup and Testing.

- Access control and security within the database system itself, including the users and databases granted with access to the databases, views and stored programs in use within the database. For more information, see Chapter 4, Access Control and Account Management.

- The features offered by security-related plugins. See Chapter 6, Security Components and Plugins.

- Network security of MySQL and your system. The security is related to the grants for individual users, but you may also wish to restrict MySQL so that it is available only locally on the MySQL server host, or to a limited set of other hosts.

- Ensure that you have adequate and appropriate backups of your database files, configuration and log files. Also be sure that you have a recovery solution in place and test that you are able to successfully recover the information from your backups. See Backup and Recovery.
Chapter 2 General Security Issues

Table of Contents

2.1 Security Guidelines ................................................................. 3
2.2 Keeping Passwords Secure ....................................................... 4
  2.2.1 End-User Guidelines for Password Security .............................. 4
  2.2.2 Administrator Guidelines for Password Security ......................... 6
  2.2.3 Passwords and Logging ...................................................... 6
2.3 Making MySQL Secure Against Attackers .................................... 7
2.4 Security-Related mysqld Options and Variables .......................... 9
2.5 How to Run MySQL as a Normal User ......................................... 9
2.6 Security Considerations for LOAD DATA LOCAL .......................... 10
2.7 Client Programming Security Guidelines ..................................... 13

This section describes general security issues to be aware of and what you can do to make your MySQL installation more secure against attack or misuse. For information specifically about the access control system that MySQL uses for setting up user accounts and checking database access, see Chapter 3, Postinstallation Setup and Testing.

For answers to some questions that are often asked about MySQL Server security issues, see Appendix A, MySQL 8.0 FAQ: Security.

2.1 Security Guidelines

Anyone using MySQL on a computer connected to the Internet should read this section to avoid the most common security mistakes.

In discussing security, it is necessary to consider fully protecting the entire server host (not just the MySQL server) against all types of applicable attacks: eavesdropping, altering, playback, and denial of service. We do not cover all aspects of availability and fault tolerance here.

MySQL uses security based on Access Control Lists (ACLs) for all connections, queries, and other operations that users can attempt to perform. There is also support for SSL-encrypted connections between MySQL clients and servers. Many of the concepts discussed here are not specific to MySQL at all; the same general ideas apply to almost all applications.

When running MySQL, follow these guidelines:

• Do not ever give anyone (except MySQL root accounts) access to the user table in the mysql system database! This is critical.

• Learn how the MySQL access privilege system works (see Chapter 4, Access Control and Account Management). Use the GRANT and REVOKE statements to control access to MySQL. Do not grant more privileges than necessary. Never grant privileges to all hosts.

  Checklist:

  • Try mysql -u root. If you are able to connect successfully to the server without being asked for a password, anyone can connect to your MySQL server as the MySQL root user with full privileges! Review the MySQL installation instructions, paying particular attention to the information about setting a root password. See Section 3.4, “Securing the Initial MySQL Account”.

  • Use the SHOW GRANTS statement to check which accounts have access to what. Then use the REVOKE statement to remove those privileges that are not necessary.

  • Do not store cleartext passwords in your database. If your computer becomes compromised, the intruder can take the full list of passwords and use them. Instead, use SHA2() or some other one-way hashing function and store the hash value.
To prevent password recovery using rainbow tables, do not use these functions on a plain password; instead, choose some string to be used as a salt, and use hash(hash(password)+salt) values.

- Do not choose passwords from dictionaries. Special programs exist to break passwords. Even passwords like “xfish98” are very bad. Much better is “dug98” which contains the same word “fish” but typed one key to the left on a standard QWERTY keyboard. Another method is to use a password that is taken from the first characters of each word in a sentence (for example, “Four score and seven years ago” results in a password of “Fsasya”). The password is easy to remember and type, but difficult to guess for someone who does not know the sentence. In this case, you can additionally substitute digits for the number words to obtain the phrase “4 score and 7 years ago”, yielding the password “4sa7ya” which is even more difficult to guess.

- Invest in a firewall. This protects you from at least 50% of all types of exploits in any software. Put MySQL behind the firewall or in a demilitarized zone (DMZ).

Checklist:

- Try to scan your ports from the Internet using a tool such as nmap. MySQL uses port 3306 by default. This port should not be accessible from untrusted hosts. As a simple way to check whether your MySQL port is open, try the following command from some remote machine, where server_host is the host name or IP address of the host on which your MySQL server runs:

  shell> telnet server_host 3306

  If telnet hangs or the connection is refused, the port is blocked, which is how you want it to be. If you get a connection and some garbage characters, the port is open, and should be closed on your firewall or router, unless you really have a good reason to keep it open.

- Applications that access MySQL should not trust any data entered by users, and should be written using proper defensive programming techniques. See Section 2.7, “Client Programming Security Guidelines”.

- Do not transmit plain (unencrypted) data over the Internet. This information is accessible to everyone who has the time and ability to intercept it and use it for their own purposes. Instead, use an encrypted protocol such as SSL or SSH. MySQL supports internal SSL connections. Another technique is to use SSH port-forwarding to create an encrypted (and compressed) tunnel for the communication.

- Learn to use the tcpdump and strings utilities. In most cases, you can check whether MySQL data streams are unencrypted by issuing a command like the following:

  shell> tcpdump -l -i eth0 -w - src or dst port 3306 | strings

  This works under Linux and should work with small modifications under other systems.

  Warning

  If you do not see cleartext data, this does not always mean that the information actually is encrypted. If you need high security, consult with a security expert.

2.2 Keeping Passwords Secure

Passwords occur in several contexts within MySQL. The following sections provide guidelines that enable end users and administrators to keep these passwords secure and avoid exposing them. In addition, the validate_password plugin can be used to enforce a policy on acceptable password. See Section 6.3, “The Password Validation Component”.

2.2.1 End-User Guidelines for Password Security
MySQL users should use the following guidelines to keep passwords secure.

When you run a client program to connect to the MySQL server, it is inadvisable to specify your password in a way that exposes it to discovery by other users. The methods you can use to specify your password when you run client programs are listed here, along with an assessment of the risks of each method. In short, the safest methods are to have the client program prompt for the password or to specify the password in a properly protected option file.

- **Use the `mysql_config_editor` utility**, which enables you to store authentication credentials in an encrypted login path file named `.mylogin.cnf`. The file can be read later by MySQL client programs to obtain authentication credentials for connecting to MySQL Server. See `mysql_config_editor — MySQL Configuration Utility`.

- **Use a `--password=password` or `-p` option** on the command line. For example:
  
  ```shell
  shell> mysql -u francis -pfrank db_name
  ```

  **Warning**

  This is convenient *but insecure*. On some systems, your password becomes visible to system status programs such as `ps` that may be invoked by other users to display command lines. MySQL clients typically overwrite the command-line password argument with zeros during their initialization sequence. However, there is still a brief interval during which the value is visible. Also, on some systems this overwriting strategy is ineffective and the password remains visible to `ps`. (SystemV Unix systems and perhaps others are subject to this problem.)

- **Use the `--password` or `-p` option** on the command line with no password value specified. In this case, the client program solicits the password interactively:

  ```shell
  shell> mysql -u francis -p db_name
  Enter password: ********
  ```

  The `*` characters indicate where you enter your password. The password is not displayed as you enter it.

  It is more secure to enter your password this way than to specify it on the command line because it is not visible to other users. However, this method of entering a password is suitable only for programs that you run interactively. If you want to invoke a client from a script that runs noninteractively, there is no opportunity to enter the password from the keyboard. On some systems, you may even find that the first line of your script is read and interpreted (incorrectly) as your password.

- **Store your password in an option file**. For example, on Unix, you can list your password in the `[client]` section of the `.my.cnf` file in your home directory:

  ```ini
  [client]
  password=password
  ```

  To keep the password safe, the file should not be accessible to anyone but yourself. To ensure this, set the file access mode to `400` or `600`. For example:

  ```shell
  shell> chmod 600 .my.cnf
  ```

  To name from the command line a specific option file containing the password, use the `--defaults-file=file_name` option, where `file_name` is the full path name to the file. For example:

  ```shell
  shell> mysql --defaults-file=/home/francis/mysql-opts
  ```
Using Option Files, discusses option files in more detail.

On Unix, the mysql client writes a record of executed statements to a history file (see mysql Client Logging). By default, this file is named .mysql_history and is created in your home directory. Passwords can be written as plain text in SQL statements such as CREATE USER and ALTER USER, so if you use these statements, they are logged in the history file. To keep this file safe, use a restrictive access mode, the same way as described earlier for the .my.cnf file.

If your command interpreter is configured to maintain a history, any file in which the commands are saved will contain MySQL passwords entered on the command line. For example, bash uses ~/.bash_history. Any such file should have a restrictive access mode.

2.2.2 Administrator Guidelines for Password Security

Database administrators should use the following guidelines to keep passwords secure.

MySQL stores passwords for user accounts in the mysql.user system table. Access to this table should never be granted to any nonadministrative accounts.

Account passwords can be expired so that users must reset them. See Section 4.15, ”Password Management”, and Section 4.16, ”Server Handling of Expired Passwords”.

The validate_password plugin can be used to enforce a policy on acceptable password. See Section 6.3, ”The Password Validation Component”.

A user who has access to modify the plugin directory (the value of the plugin_dir system variable) or the my.cnf file that specifies the plugin directory location can replace plugins and modify the capabilities provided by plugins, including authentication plugins.

Files such as log files to which passwords might be written should be protected. See Section 2.2.3, ”Passwords and Logging”.

2.2.3 Passwords and Logging

Passwords can be written as plain text in SQL statements such as CREATE USER, GRANT and SET PASSWORD. If such statements are logged by the MySQL server as written, passwords in them become visible to anyone with access to the logs.

Statement logging avoids writing passwords as cleartext for the following statements:

```
CREATE USER ... IDENTIFIED BY ...
ALTER USER ... IDENTIFIED BY ...
SET PASSWORD ...
SLAVE START ... PASSWORD = ...
CREATE SERVER ... OPTIONS(... PASSWORD ...) 
ALTER SERVER ... OPTIONS(... PASSWORD ...) 
```

Passwords in those statements are rewritten to not appear literally in statement text written to the general query log, slow query log, and binary log. Rewriting does not apply to other statements. In particular, INSERT or UPDATE statements for the mysql.user system table that refer to literal passwords are logged as is, so you should avoid such statements. (Direct modification of grant tables is discouraged, anyway.)

For the general query log, password rewriting can be suppressed by starting the server with the --log-raw option. For security reasons, this option is not recommended for production use. For diagnostic purposes, it may be useful to see the exact text of statements as received by the server.

By default, contents of audit log files produced by the audit log plugin are not encrypted and may contain sensitive information, such as the text of SQL statements. For security reasons, audit log files
Making MySQL Secure Against Attackers

should be written to a directory accessible only to the MySQL server and to users with a legitimate reason to view the log. See Section 6.5.3, “MySQL Enterprise Audit Security Considerations”.

Statements received by the server may be rewritten if a query rewrite plugin is installed (see Query Rewrite Plugins). In this case, the \--log-raw option affects statement logging as follows:

\- Without \--log-raw, the server logs the statement returned by the query rewrite plugin. This may differ from the statement as received.

\- With \--log-raw, the server logs the original statement as received.

An implication of password rewriting is that statements that cannot be parsed (due, for example, to syntax errors) are not written to the general query log because they cannot be known to be password free. Use cases that require logging of all statements including those with errors should use the \--log-raw option, bearing in mind that this also bypasses password rewriting.

Password rewriting occurs only when plain text passwords are expected. For statements with syntax that expect a password hash value, no rewriting occurs. If a plain text password is supplied erroneously for such syntax, the password is logged as given, without rewriting.

To guard log files against unwarranted exposure, locate them in a directory that restricts access to the server and the database administrator. If the server logs to tables in the \texttt{mysql} database, grant access to those tables only to the database administrator.

Replication slaves store the password for the replication master in the master info repository, which by default is a table in the \texttt{mysql} database named \texttt{slave_master_info}. The use of a file in the data directory for the master info repository is now deprecated, but still possible (see Replication Relay and Status Logs). Ensure that the master info repository can be accessed only by the database administrator. An alternative to storing the password in the master info repository is to use the \texttt{START SLAVE} statement to specify credentials for connecting to the master.

Use a restricted access mode to protect database backups that include log tables or log files containing passwords.

\section*{2.3 Making MySQL Secure Against Attackers}

When you connect to a MySQL server, you should use a password. The password is not transmitted as cleartext over the connection.

All other information is transferred as text, and can be read by anyone who is able to watch the connection. If the connection between the client and the server goes through an untrusted network, and you are concerned about this, you can use the compressed protocol to make traffic much more difficult to decipher. You can also use MySQL's internal SSL support to make the connection even more secure. See Chapter 5, Using Encrypted Connections. Alternatively, use SSH to get an encrypted TCP/IP connection between a MySQL server and a MySQL client. You can find an Open Source SSH client at \url{http://www.openssh.org/}, and a comparison of both Open Source and Commercial SSH clients at \url{http://en.wikipedia.org/wiki/Comparison_of_SSH_clients}.

To make a MySQL system secure, you should strongly consider the following suggestions:

\- Require all MySQL accounts to have a password. A client program does not necessarily know the identity of the person running it. It is common for client/server applications that the user can specify any user name to the client program. For example, anyone can use the \texttt{mysql} program to connect as any other person simply by invoking it as \texttt{mysql -u other_user db_name} if \texttt{other_user} has no password. If all accounts have a password, connecting using another user's account becomes much more difficult.

For a discussion of methods for setting passwords, see Section 4.14, “Assigning Account Passwords”.

7
Making MySQL Secure Against Attackers

- Make sure that the only Unix user account with read or write privileges in the database directories is the account that is used for running `mysqld`.

- Never run the MySQL server as the Unix `root` user. This is extremely dangerous, because any user with the `FILE` privilege is able to cause the server to create files as `root` (for example, `~root/.bashrc`). To prevent this, `mysqld` refuses to run as `root` unless that is specified explicitly using the `--user=root` option.

  `mysqld` can (and should) be run as an ordinary, unprivileged user instead. You can create a separate Unix account named `mysql` to make everything even more secure. Use this account only for administering MySQL. To start `mysqld` as a different Unix user, add a `user` option that specifies the user name in the `[mysqld]` group of the `my.cnf` option file where you specify server options. For example:

  ```
  [mysqld]
  user=mysql
  ```

  This causes the server to start as the designated user whether you start it manually or by using `mysqld_safe` or `mysql.server`. For more details, see Section 2.5, “How to Run MySQL as a Normal User”.

  Running `mysqld` as a Unix user other than `root` does not mean that you need to change the `root` user name in the `user` table. User names for MySQL accounts have nothing to do with user names for Unix accounts.

- Do not grant the `FILE` privilege to nonadministrative users. Any user that has this privilege can write a file anywhere in the file system with the privileges of the `mysqld` daemon. This includes the server's data directory containing the files that implement the privilege tables. To make `FILE`-privilege operations a bit safer, files generated with `SELECT ... INTO OUTFILE` do not overwrite existing files and are writable by everyone.

  The `FILE` privilege may also be used to read any file that is world-readable or accessible to the Unix user that the server runs as. With this privilege, you can read any file into a database table. This could be abused, for example, by using `LOAD DATA` to load `/etc/passwd` into a table, which then can be displayed with `SELECT`.

  To limit the location in which files can be read and written, set the `secure_file_priv` system to a specific directory. See Server System Variables.

- Encrypt binary log files and relay log files. Encryption helps to protect these files and the potentially sensitive data contained in them from being misused by outside attackers, and also from unauthorized viewing by users of the operating system where they are stored. You enable encryption on a MySQL server by setting the `binlog_encryption` system variable to `ON`. For more information, see Encrypting Binary Log Files and Relay Log Files.

- Do not grant the `PROCESS` or `SUPER` privilege to nonadministrative users. The output of `mysqladmin processlist` and `SHOW PROCESSLIST` shows the text of any statements currently being executed, so any user who is permitted to see the server process list might be able to see statements issued by other users.

  `mysqld` reserves an extra connection for users who have the `CONNECTION_ADMIN` or `SUPER` privilege, so that a MySQL `root` user can log in and check server activity even if all normal connections are in use.

  The `SUPER` privilege can be used to terminate client connections, change server operation by changing the value of system variables, and control replication servers.

- Do not permit the use of symlinks to tables. (This capability can be disabled with the `--skip-symbolic-links` option.) This is especially important if you run `mysqld` as `root`, because anyone that has write access to the server's data directory then could delete any file in the system! See Using Symbolic Links for MyISAM Tables on Unix.
Security-Related mysqld Options and Variables

- Stored programs and views should be written using the security guidelines discussed in Stored Object Access Control.

- If you do not trust your DNS, you should use IP addresses rather than host names in the grant tables. In any case, you should be very careful about creating grant table entries using host name values that contain wildcards.

- If you want to restrict the number of connections permitted to a single account, you can do so by setting the `max_user_connections` variable in `mysqld`. The `CREATE USER` and `ALTER USER` statements also support resource control options for limiting the extent of server use permitted to an account. See CREATE USER Statement, and ALTER USER Statement.

- If the plugin directory is writable by the server, it may be possible for a user to write executable code to a file in the directory using `SELECT ... INTO DUMPFILE`. This can be prevented by making `plugin_dir` read only to the server or by setting `secure_file_priv` to a directory where `SELECT` writes can be made safely.

### 2.4 Security-Related mysqld Options and Variables

The following table shows `mysqld` options and system variables that affect security. For descriptions of each of these, see Server Command Options, and Server System Variables.

<table>
<thead>
<tr>
<th>Name</th>
<th>Cmd-Line</th>
<th>Option File</th>
<th>System Var</th>
<th>Status Var</th>
<th>Var Scope</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow-suspicious-udfs</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td>automatic_sp_privileges</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td>chroot</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td>des-key-file</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td>local_infile</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>Both</td>
<td>Yes</td>
</tr>
<tr>
<td>old_passwords</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>Both</td>
<td>Yes</td>
</tr>
<tr>
<td>safe-user-create</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Both</td>
<td>Yes</td>
</tr>
<tr>
<td>secure_auth</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td>secure_file_priv</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>Global</td>
<td>No</td>
</tr>
<tr>
<td>skip-grant-tables</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Global</td>
<td>No</td>
</tr>
<tr>
<td>skip_name_resolve</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>Global</td>
<td>No</td>
</tr>
<tr>
<td>skip_networking</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>Global</td>
<td>No</td>
</tr>
<tr>
<td>skip_show_database</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>Global</td>
<td>No</td>
</tr>
</tbody>
</table>

### 2.5 How to Run MySQL as a Normal User

On Windows, you can run the server as a Windows service using a normal user account.

On Linux, for installations performed using a MySQL repository or RPM packages, the MySQL server `mysqld` should be started by the local `mysql` operating system user. Starting by another operating system user is not supported by the init scripts that are included as part of the MySQL repositories.

On Unix (or Linux for installations performed using `tar.gz` packages), the MySQL server `mysqld` can be started and run by any user. However, you should avoid running the server as the Unix `root` user for security reasons. To change `mysqld` to run as a normal unprivileged Unix user `user_name`, you must do the following:

1. Stop the server if it is running (use `mysqladmin shutdown`).
2. Change the database directories and files so that user_name has privileges to read and write files in them (you might need to do this as the Unix root user):

```shell
shell> chown -R user_name /path/to/mysql/datadir
```

If you do not do this, the server will not be able to access databases or tables when it runs as user_name.

If directories or files within the MySQL data directory are symbolic links, chown -R might not follow symbolic links for you. If it does not, you will also need to follow those links and change the directories and files they point to.

3. Start the server as user user_name. Another alternative is to start `mysqld` as the Unix root user and use the `--user=user_name` option. `mysqld` starts, then switches to run as the Unix user user_name before accepting any connections.

4. To start the server as the given user automatically at system startup time, specify the user name by adding a user option to the [mysqld] group of the /etc/my.cnf option file or the my.cnf option file in the server's data directory. For example:

```
[mysqld]
user=user_name
```

If your Unix machine itself is not secured, you should assign passwords to the MySQL root account in the grant tables. Otherwise, any user with a login account on that machine can run the `mysql` client with a `--user=root` option and perform any operation. (It is a good idea to assign passwords to MySQL accounts in any case, but especially so when other login accounts exist on the server host.) See Section 3.4, “Securing the Initial MySQL Account”.

### 2.6 Security Considerations for LOAD DATA LOCAL

The **LOAD DATA** statement loads a data file into a table. The statement can load a file located on the server host, or, if the **LOCAL** keyword is specified, on the client host.

The **LOCAL** version of **LOAD DATA** has two potential security issues:

- Because **LOAD DATA LOCAL** is an SQL statement, parsing occurs on the server side, and transfer of the file from the client host to the server host is initiated by the MySQL server, which tells the client the file named in the statement. In theory, a patched server could tell the client program to transfer a file of the server's choosing rather than the file named in the statement. Such a server could access any file on the client host to which the client user has read access. (A patched server could in fact reply with a file-transfer request to any statement, not just **LOAD DATA LOCAL**, so a more fundamental issue is that clients should not connect to untrusted servers.)

- In a Web environment where the clients are connecting from a Web server, a user could use **LOAD DATA LOCAL** to read any files that the Web server process has read access to (assuming that a user could run any statement against the SQL server). In this environment, the client with respect to the MySQL server actually is the Web server, not a remote program being run by users who connect to the Web server.

To avoid connecting to untrusted servers, clients can establish a secure connection and verify the server identity by connecting using the `--ssl-mode=VERIFY_IDENTITY` option and the appropriate CA certificate.

To avoid **LOAD DATA** issues, clients should avoid using **LOCAL** unless proper client-side precautions have been taken.

For control over local data loading, MySQL permits the capability to be enabled or disabled. In addition, as of MySQL 8.0.21, MySQL enables clients to restrict local data loading operations to files located in a designated directory.

- **Enabling or Disabling Local Data Loading Capability**
Enabling or Disabling Local Data Loading Capability

Restricting Files Permitted for Local Data Loading

Enabling or Disabling Local Data Loading Capability

Administrators and applications can configure whether to permit local data loading as follows:

- On the server side:
  - The `local_infile` system variable controls server-side LOCAL capability. Depending on the `local_infile` setting, the server refuses or permits local data loading by clients that request local data loading.
  - By default, `local_infile` is disabled. To explicitly cause the server to refuse or permit LOAD DATA LOCAL statements (regardless of how client programs and libraries are configured at build time or runtime), start `mysqld` with `local_infile` disabled or enabled. `local_infile` can also be set at runtime.

- On the client side:
  - The `ENABLED_LOCAL_INFILE` CMake option controls the compiled-in default LOCAL capability for the MySQL client library (see MySQL Source-Configuration Options). Clients that make no explicit arrangements therefore have LOCAL capability disabled or enabled according to the `ENABLED_LOCAL_INFILE` setting specified at MySQL build time.
  - By default, the client library in MySQL binary distributions is compiled with `ENABLED_LOCAL_INFILE` disabled. If you compile MySQL from source, configure it with `ENABLED_LOCAL_INFILE` disabled or enabled based on whether clients that make no explicit arrangements should have LOCAL capability disabled or enabled.
  - For client programs that use the C API, local data loading capability is determined by the default compiled into the MySQL client library. To enable or disable it explicitly, invoke the `mysql_options()` C API function to disable or enable the `MYSQL_OPT_LOCAL_INFILE` option. See `mysql_options()`.
  - For the `mysql` client, local data loading capability is determined by the default compiled into the MySQL client library. To disable or enable it explicitly, use the `--local-infile=0` or `--local-infile[=1]` option.
  - For the `mysqlimport` client, local data loading is not used by default. To disable or enable it explicitly, use the `--local=0` or `--local[=1]` option.
  - If you use LOAD DATA LOCAL in Perl scripts or other programs that read the [client] group from option files, you can add a `local-infile` option setting to that group. To prevent problems for programs that do not understand this option, specify it using the `loose-` prefix:

```
[client]
loose-local-infile=0
```

or:

```
[client]
loose-local-infile=1
```

- In all cases, successful use of a LOCAL load operation by a client also requires that the server permits local loading.

If LOCAL capability is disabled, on either the server or client side, a client that attempts to issue a LOAD DATA LOCAL statement receives the following error message:

```
ERROR 3950 (42000): Loading local data is disabled; this must be enabled on both the client and server side
```
Restricting Files Permitted for Local Data Loading

As of MySQL 8.0.21, the MySQL client library enables client applications to restrict local data loading operations to files located in a designated directory. Certain MySQL client programs take advantage of this capability.

Client programs that use the C API can control which files to permit for load data loading using the MYSQL_OPT_LOCAL_INFILE and MYSQL_OPT_LOAD_DATA_LOCAL_DIR options of the mysql_options() C API function (see mysql_options()).

The effect of MYSQL_OPT_LOAD_DATA_LOCAL_DIR depends on whether LOCAL data loading is enabled or disabled:

- If LOCAL data loading is enabled, either by default in the MySQL client library or by explicitly enabling MYSQL_OPT_LOCAL_INFILE, the MYSQL_OPT_LOAD_DATA_LOCAL_DIR option has no effect.

- If LOCAL data loading is disabled, either by default in the MySQL client library or by explicitly disabling MYSQL_OPT_LOCAL_INFILE, the MYSQL_OPT_LOAD_DATA_LOCAL_DIR option can be used to designate a permitted directory for locally loaded files. In this case, LOCAL data loading is permitted but restricted to files located in the designated directory. Interpretation of the MYSQL_OPT_LOAD_DATA_LOCAL_DIR value is as follows:
  - If the value is the null pointer (the default), it names no directory, with the result that no files are permitted for LOCAL data loading.
  
  - If the value is a directory path name, LOCAL data loading is permitted but restricted to files located in the named directory. Comparison of the directory path name and the path name of files to be loaded is case-sensitive regardless of the case-sensitivity of the underlying file system.

MySQL client programs use the preceding mysql_options() options as follows:

- The mysql client has a --load-data-local-dir option that takes a directory path or an empty string. mysql uses the option value to set the MYSQL_OPT_LOAD_DATA_LOCAL_DIR option (with an empty string setting the value to the null pointer). The effect of --load-data-local-dir depends on whether LOCAL data loading is enabled:
  - If LOCAL data loading is enabled, either by default in the MySQL client library or by specifying --local-infile[=1], the --load-data-local-dir option is ignored.
  
  - If LOCAL data loading is disabled, either by default in the MySQL client library or by specifying --local-infile=0, the --load-data-local-dir option applies.

When --load-data-local-dir applies, the option value designates the directory in which local data files must be located. Comparison of the directory path name and the path name of files to be loaded is case-sensitive regardless of the case-sensitivity of the underlying file system. If the option value is the empty string, it names no directory, with the result that no files are permitted for local data loading.

- mysqlimport sets MYSQL_OPT_LOAD_DATA_LOCAL_DIR for each file that it processes so that the directory containing the file is the permitted local loading directory.

- For data loading operations corresponding to LOAD DATA statements, mysqlbinlog extracts the files from the binary log events, writes them as temporary files to the local file system, and writes LOAD DATA LOCAL statements to cause the files to be loaded. By default, mysqlbinlog writes these temporary files to an operating system-specific directory. The --local-load option can be used to explicitly specify the directory where mysqlbinlog should prepare local temporary files.

Because other processes can write files to the default system-specific directory, it is advisable to specify the --local-load option to mysqlbinlog to designate a different directory for data files, and then designate that same directory by specifying the --load-data-local-dir option to mysql when processing the output from mysqlbinlog.
2.7 Client Programming Security Guidelines

Applications that access MySQL should not trust any data entered by users, who can try to trick your code by entering special or escaped character sequences in Web forms, URLs, or whatever application you have built. Be sure that your application remains secure if a user enters something like `DROP DATABASE mysql;`. This is an extreme example, but large security leaks and data loss might occur as a result of hackers using similar techniques, if you do not prepare for them.

A common mistake is to protect only string data values. Remember to check numeric data as well. If an application generates a query such as `SELECT * FROM table WHERE ID=234` when a user enters the value 234, the user can enter the value 234 OR 1=1 to cause the application to generate the query `SELECT * FROM table WHERE ID=234 OR 1=1`. As a result, the server retrieves every row in the table. This exposes every row and causes excessive server load. The simplest way to protect from this type of attack is to use single quotation marks around the numeric constants: `SELECT * FROM table WHERE ID='234'`. If the user enters extra information, it all becomes part of the string. In a numeric context, MySQL automatically converts this string to a number and strips any trailing nonnumeric characters from it.

Sometimes people think that if a database contains only publicly available data, it need not be protected. This is incorrect. Even if it is permissible to display any row in the database, you should still protect against denial of service attacks (for example, those that are based on the technique in the preceding paragraph that causes the server to waste resources). Otherwise, your server becomes unresponsive to legitimate users.

Checklist:

• Enable strict SQL mode to tell the server to be more restrictive of what data values it accepts. See Server SQL Modes.

• Try to enter single and double quotation marks (‘ and ”) in all of your Web forms. If you get any kind of MySQL error, investigate the problem right away.

• Try to modify dynamic URLs by adding %22 ("), %23 (#), and %27 (‘) to them.

• Try to modify data types in dynamic URLs from numeric to character types using the characters shown in the previous examples. Your application should be safe against these and similar attacks.

• Try to enter characters, spaces, and special symbols rather than numbers in numeric fields. Your application should remove them before passing them to MySQL or else generate an error. Passing unchecked values to MySQL is very dangerous!

• Check the size of data before passing it to MySQL.

• Have your application connect to the database using a user name different from the one you use for administrative purposes. Do not give your applications any access privileges they do not need.

Many application programming interfaces provide a means of escaping special characters in data values. Properly used, this prevents application users from entering values that cause the application to generate statements that have a different effect than you intend:

• MySQL C API: Use the `mysql_real_escape_string_quote()` API call.

• MySQL++: Use the escape and quote modifiers for query streams.

• PHP: Use either the `mysqli` or `pdo_mysql` extensions, and not the older `ext/mysql` extension. The preferred API’s support the improved MySQL authentication protocol and passwords, as well as prepared statements with placeholders. See also Choosing an API.

If the older `ext/mysql` extension must be used, then for escaping use the `mysql_real_escape_string_quote()` function and not `mysql_escape_string()` or
addslashes() because only mysql_real_escape_string_quote() is character set-aware; the other functions can be “bypassed” when using (invalid) multibyte character sets.

• Perl DBI: Use placeholders or the quote() method.
• Ruby DBI: Use placeholders or the quote() method.
• Java JDBC: Use a PreparedStatement object and placeholders.

Other programming interfaces might have similar capabilities.
Chapter 3 Postinstallation Setup and Testing

Table of Contents

3.1 Initializing the Data Directory ................................. 15
3.2 Starting the Server ................................................ 20
    3.2.1 Troubleshooting Problems Starting the MySQL Server ........................................ 21
3.3 Testing the Server ................................................ 23
3.4 Securing the Initial MySQL Account .......................... 25
3.5 Starting and Stopping MySQL Automatically .................. 27

This section discusses tasks that you should perform after installing MySQL:

- If necessary, initialize the data directory and create the MySQL grant tables. For some MySQL installation methods, data directory initialization may be done for you automatically:
  - Windows installation operations performed by MySQL Installer.
  - Installation on Linux using a server RPM or Debian distribution from Oracle.
  - Installation using the native packaging system on many platforms, including Debian Linux, Ubuntu Linux, Gentoo Linux, and others.
  - Installation on macOS using a DMG distribution.

For other platforms and installation types, you must initialize the data directory manually. These include installation from generic binary and source distributions on Unix and Unix-like systems, and installation from a ZIP Archive package on Windows. For instructions, see Section 3.1, “Initializing the Data Directory”.

- Start the server and make sure that it can be accessed. For instructions, see Section 3.2, “Starting the Server”, and Section 3.3, “Testing the Server”.

- Assign passwords to the initial root account in the grant tables, if that was not already done during data directory initialization. Passwords prevent unauthorized access to the MySQL server. For instructions, see Section 3.4, “Securing the Initial MySQL Account”.

- Optionally, arrange for the server to start and stop automatically when your system starts and stops. For instructions, see Section 3.5, “Starting and Stopping MySQL Automatically”.

- Optionally, populate time zone tables to enable recognition of named time zones. For instructions, see MySQL Server Time Zone Support.

When you are ready to create additional user accounts, you can find information on the MySQL access control system and account management in Chapter 4, Access Control and Account Management.

3.1 Initializing the Data Directory

After MySQL is installed, the data directory must be initialized, including the tables in the mysql system schema:

- For some MySQL installation methods, data directory initialization is automatic, as described in Chapter 3, Postinstallation Setup and Testing.

- For other installation methods, you must initialize the data directory manually. These include installation from generic binary and source distributions on Unix and Unix-like systems, and installation from a ZIP Archive package on Windows.
This section describes how to initialize the data directory manually for MySQL installation methods for which data directory initialization is not automatic. For some suggested commands that enable testing whether the server is accessible and working properly, see Section 3.3, “Testing the Server”.

**Note**

In MySQL 8.0, the default authentication plugin has changed from `mysql_native_password` to `caching_sha2_password`, and the `'root'@'localhost'` administrative account uses `caching_sha2_password` by default. If you prefer that the `root` account use the previous default authentication plugin (`mysql_native_password`), see `caching_sha2_password` and the root Administrative Account.

- Data Directory Initialization Overview
- Data Directory Initialization Procedure
- Server Actions During Data Directory Initialization
- Post-Initialization root Password Assignment

## Data Directory Initialization Overview

In the examples shown here, the server is intended to run under the user ID of the `mysql` login account. Either create the account if it does not exist (see Create a mysql User and Group), or substitute the name of a different existing login account that you plan to use for running the server.

1. Change location to the top-level directory of your MySQL installation, which is typically `/usr/local/mysql` (adjust the path name for your system as necessary):

```bash
cd /usr/local/mysql
```

Within the directory you will find several files and subdirectories, including the `bin` subdirectory that contains the server as well as client and utility programs.

2. The `secure_file_priv` system variable limits import and export operations to a specific directory. Create a directory whose location can be specified as the value of that variable:

```bash
mkdir mysql-files
```

Grant directory user and group ownership to the `mysql` user and `mysql` group, and set the directory permissions appropriately:

```bash
chown mysql:mysql mysql-files
chmod 750 mysql-files
```

3. Use the server to initialize the data directory, including the `mysql` schema containing the initial MySQL grant tables that determine how users are permitted to connect to the server. For example:

```bash
bin/mysqld --initialize --user=mysql
```

For important information about the command, especially regarding command options you might use, see Data Directory Initialization Procedure. For details about how the server performs initialization, see Server Actions During Data Directory Initialization.

Typically, data directory initialization need be done only after you first install MySQL. (For upgrades to an existing installation, perform the upgrade procedure instead; see Upgrading MySQL.) However, the command that initializes the data directory does not overwrite any existing `mysql` schema tables, so it is safe to run in any circumstances.

4. If you want to deploy the server with automatic support for secure connections, use the `mysql_ssl_rsa_setup` utility to create default SSL and RSA files:
Data Directory Initialization Procedure

Change location to the top-level directory of your MySQL installation, which is typically /usr/local/mysql (adjust the path name for your system as necessary):

\[ cd /usr/local/mysql \]

To initialize the data directory, invoke mysqld with the `--initialize` or `--initialize-insecure` option, depending on whether you want the server to generate a random initial password for the `root'@'localhost` account, or to create that account with no password:

- Use `--initialize` for “secure by default” installation (that is, including generation of a random initial root password). In this case, the password is marked as expired and you will need to choose a new one.
- With `--initialize-insecure`, no root password is generated. This is insecure; it is assumed that you will assign a password to the account in timely fashion before putting the server into production use.

For instructions on assigning a new `root'@'localhost` password, see Post-Initialization root Password Assignment.

**Note**

The server writes any messages (including any initial password) to its standard error output. This may be redirected to the error log, so look there if you do not see the messages on your screen. For information about the error log, including where it is located, see The Error Log.

On Windows, use the `--console` option to direct messages to the console.

On Unix and Unix-like systems, it is important for the database directories and files to be owned by the mysql login account so that the server has read and write access to them when you run it later. To ensure this, start mysqld from the system root account and include the `--user` option as shown here:

\[ bin/mysqld --initialize --user=mysql \]
\[ bin/mysqld --initialize-insecure --user=mysql \]

Alternatively, execute mysql while logged in as mysql, in which case you can omit the `--user` option from the command.

On Windows, use one of these commands:

\[ bin\mysqld --initialize --console \]
\[ bin\mysqld --initialize-insecure --console \]
Note
Data directory initialization might fail if required system libraries are missing. For example, you might see an error like this:

```bash
bin/mysqld: error while loading shared libraries:
libnuma.so.1: cannot open shared object file:
No such file or directory
```

If this happens, you must install the missing libraries manually or with your system's package manager. Then retry the data directory initialization command.

It might be necessary to specify other options such as `--basedir` or `--datadir` if `mysqld` cannot identify the correct locations for the installation directory or data directory. For example (enter the command on a single line):

```bash
bin/mysqld --initialize --user=mysql
   --basedir=/opt/mysql/mysql
   --datadir=/opt/mysql/mysql/data
```

Alternatively, put the relevant option settings in an option file and pass the name of that file to `mysqld`. For Unix and Unix-like systems, suppose that the option file name is `/opt/mysql/mysql/etc/my.cnf`. Put these lines in the file:

```ini
[mysqld]
basedir=/opt/mysql/mysql
datadir=/opt/mysql/mysql/data
```

Then invoke `mysqld` as follows (enter the command on a single line with the `--defaults-file` option first):

```bash
bin/mysqld --defaults-file=/opt/mysql/mysql/etc/my.cnf
   --initialize --user=mysql
```

On Windows, suppose that `C:\my.ini` contains these lines:

```ini
[mysqld]
basedir=C:\Program Files\MySQL\MySQL Server 8.0
datadir=D:\MySQLdata
```

Then invoke `mysqld` as follows (enter the command on a single line with the `--defaults-file` option first):

```bash
bin\mysqld --defaults-file=C:\my.ini
   --initialize --console
```

Server Actions During Data Directory Initialization

Note
The data directory initialization sequence performed by the server does not substitute for the actions performed by `mysql_secure_installation` and `mysql_ssl_rsa_setup`. See `mysql_secure_installation` — Improve MySQL Installation Security, and `mysql_ssl_rsa_setup` — Create SSL/RSA Files.

When invoked with the `--initialize` or `--initialize-insecure` option, `mysqld` performs the following actions during the data directory initialization sequence:

1. The server checks for the existence of the data directory as follows:
   - If no data directory exists, the server creates it.
   - If the data directory exists but is not empty (that is, it contains files or subdirectories), the server exits after producing an error message:
Server Actions During Data Directory Initialization

[ERROR] --initialize specified but the data directory exists. Aborting.

In this case, remove or rename the data directory and try again.

An existing data directory is permitted to be nonempty if every entry has a name that begins with a period (..).

2. Within the data directory, the server creates the `mysql` system schema and its tables, including the data dictionary tables, grant tables, time zone tables, and server-side help tables. See The mysql System Schema.

3. The server initializes the `system tablespace` and related data structures needed to manage InnoDB tables.

**Note**

After `mysqld` sets up the InnoDB system tablespace, certain changes to tablespace characteristics require setting up a whole new instance. Qualifying changes include the file name of the first file in the system tablespace and the number of undo logs. If you do not want to use the default values, make sure that the settings for the `innodb_data_file_path` and `innodb_log_file_size` configuration parameters are in place in the MySQL configuration file before running `mysqld`. Also make sure to specify as necessary other parameters that affect the creation and location of InnoDB files, such as `innodb_data_home_dir` and `innodb_log_group_home_dir`.

If those options are in your configuration file but that file is not in a location that MySQL reads by default, specify the file location using the `--defaults-extra-file` option when you run `mysqld`.

4. The server creates a `root'@'localhost` superuser account and other reserved accounts (see Section 4.9, "Reserved Accounts"). Some reserved accounts are locked and cannot be used by clients, but `root'@'localhost` is intended for administrative use and you should assign it a password.

Server actions with respect to a password for the `root'@'localhost` account depend on how you invoke it:

- With `--initialize` but not `--initialize-insecure`, the server generates a random password, marks it as expired, and writes a message displaying the password:

  `[Warning] A temporary password is generated for root@localhost: iTag*AfrH5ej`

- With `--initialize-insecure`, (either with or without `--initialize` because `--initialize-insecure` implies `--initialize`), the server does not generate a password or mark it expired, and writes a warning message:

  `[Warning] root@localhost is created with an empty password ! Please consider switching off the --initialize-insecure option.`

For instructions on assigning a new `root'@'localhost` password, see Post-Initialization root Password Assignment.

5. The server populates the server-side help tables used for the `HELP` statement (see HELP Statement). The server does not populate the time zone tables. To do so manually, see MySQL Server Time Zone Support.

6. If the `init_file` system variable was given to name a file of SQL statements, the server executes the statements in the file. This option enables you to perform custom bootstrapping sequences.
When the server operates in bootstrap mode, some functionality is unavailable that limits the statements permitted in the file. These include statements that relate to account management (such as `CREATE USER` or `GRANT`), replication, and global transaction identifiers.

7. The server exits.

### Post-Initialization root Password Assignment

After you initialize the data directory by starting the server with `--initialize` or `--initialize-insecure`, start the server normally (that is, without either of those options) and assign the 'root'@'localhost' account a new password:

1. Start the server. For instructions, see Section 3.2, “Starting the Server”.
2. Connect to the server:
   - If you used `--initialize` but not `--initialize-insecure` to initialize the data directory, connect to the server as root:
     ```
     mysql -u root -p
     ```
     Then, at the password prompt, enter the random password that the server generated during the initialization sequence:
     ```
     Enter password: (enter the random root password here)
     ```
     Look in the server error log if you do not know this password.
   - If you used `--initialize-insecure` to initialize the data directory, connect to the server as root without a password:
     ```
     mysql -u root --skip-password
     ```
3. After connecting, use an `ALTER USER` statement to assign a new root password:
   ```
   ALTER USER 'root'@'localhost' IDENTIFIED BY 'root-password';
   ```
   See also Section 3.4, “Securing the Initial MySQL Account”.

**Note**
Attempts to connect to the host `127.0.0.1` normally resolve to the `localhost` account. However, this fails if the server is run with `skip_name_resolve` enabled. If you plan to do that, make sure that an account exists that can accept a connection. For example, to be able to connect as root using `--host=127.0.0.1` or `--host=::1`, create these accounts:

```
CREATE USER 'root'@'127.0.0.1' IDENTIFIED BY 'root-password';
CREATE USER 'root'@'::1' IDENTIFIED BY 'root-password';
```

It is possible to put those statements in a file to be executed using the `init_file` system variable, as discussed in Server Actions During Data Directory Initialization.

### 3.2 Starting the Server

This section describes how start the server on Unix and Unix-like systems. (For Windows, see Starting the Server for the First Time.) For some suggested commands that you can use to test whether the server is accessible and working properly, see Section 3.3, “Testing the Server”.

Start the MySQL server like this if your installation includes `mysqld_safe`:

```
shell> bin/mysqld_safe --user=mysql &
```
Troubleshooting Problems Starting the MySQL Server

**Note**

For Linux systems on which MySQL is installed using RPM packages, server startup and shutdown is managed using systemd rather than `mysqld_safe`, and `mysqld_safe` is not installed. See [Managing MySQL Server with systemd](#).

Start the server like this if your installation includes systemd support:

```shell
systemctl start mysqld
```

Substitute the appropriate service name if it differs from `mysqld` (for example, `mysql` on SLES systems).

It is important that the MySQL server be run using an unprivileged (non-root) login account. To ensure this, run `mysqld_safe` as `root` and include the `--user` option as shown. Otherwise, you should execute the program while logged in as `mysql`, in which case you can omit the `--user` option from the command.

For further instructions for running MySQL as an unprivileged user, see Section 2.5, “How to Run MySQL as a Normal User”.

If the command fails immediately and prints `mysqld ended`, look for information in the error log (which by default is the `host_name.err` file in the data directory).

If the server is unable to access the data directory it starts or read the grant tables in the `mysql` schema, it writes a message to its error log. Such problems can occur if you neglected to create the grant tables by initializing the data directory before proceeding to this step, or if you ran the command that initializes the data directory without the `--user` option. Remove the `data` directory and run the command with the `--user` option.

If you have other problems starting the server, see Section 3.2.1, “Troubleshooting Problems Starting the MySQL Server”. For more information about `mysqld_safe`, see `mysqld_safe — MySQL Server Startup Script`. For more information about systemd support, see [Managing MySQL Server with systemd](#).

### 3.2.1 Troubleshooting Problems Starting the MySQL Server

This section provides troubleshooting suggestions for problems starting the server. For additional suggestions for Windows systems, see [Troubleshooting a Microsoft Windows MySQL Server Installation](#).

If you have problems starting the server, here are some things to try:

- Check the [error log](#) to see why the server does not start. Log files are located in the [data directory](#) (typically `C:\Program Files\MySQL\MySQL Server 8.0\data` on Windows, `/usr/local/mysql/data` for a Unix/Linux binary distribution, and `/usr/local/var` for a Unix/Linux source distribution). Look in the data directory for files with names of the form `host_name.err` and `host_name.log`, where `host_name` is the name of your server host. Then examine the last few lines of these files. Use `tail` to display them:

  ```shell
  tail host_name.err
  tail host_name.log
  ```

- Specify any special options needed by the storage engines you are using. You can create a `my.cnf` file and specify startup options for the engines that you plan to use. If you are going to use storage engines that support transactional tables (`InnoDB`, `NDB`), be sure that you have them configured the way you want before starting the server. If you are using `InnoDB` tables, see [InnoDB Configuration](#) for guidelines and [InnoDB Startup Options and System Variables](#) for option syntax.

Although storage engines use default values for options that you omit, Oracle recommends that you review the available options and specify explicit values for any options whose defaults are not appropriate for your installation.
Troubleshooting Problems Starting the MySQL Server

- Make sure that the server knows where to find the data directory. The mysqld server uses this directory as its current directory. This is where it expects to find databases and where it expects to write log files. The server also writes the pid (process ID) file in the data directory.

  The default data directory location is hardcoded when the server is compiled. To determine what the default path settings are, invoke mysqld with the --verbose and --help options. If the data directory is located somewhere else on your system, specify that location with the --datadir option to mysqld or mysqld_safe, on the command line or in an option file. Otherwise, the server will not work properly. As an alternative to the --datadir option, you can specify mysqld the location of the base directory under which MySQL is installed with the --basedir, and mysqld looks for the data directory there.

  To check the effect of specifying path options, invoke mysqld with those options followed by the --verbose and --help options. For example, if you change location to the directory where mysqld is installed and then run the following command, it shows the effect of starting the server with a base directory of /usr/local:

  shell> ./mysqld --basedir=/usr/local --verbose --help

  You can specify other options such as --datadir as well, but --verbose and --help must be the last options.

  Once you determine the path settings you want, start the server without --verbose and --help.

  If mysqld is currently running, you can find out what path settings it is using by executing this command:

  shell> mysqladmin variables

  Or:

  shell> mysqladmin -h host_name variables

  host_name is the name of the MySQL server host.

- Make sure that the server can access the data directory. The ownership and permissions of the data directory and its contents must allow the server to read and modify them.

  If you get Errcode 13 (which means Permission denied) when starting mysqld, this means that the privileges of the data directory or its contents do not permit server access. In this case, you change the permissions for the involved files and directories so that the server has the right to use them. You can also start the server as root, but this raises security issues and should be avoided.

  Change location to the data directory and check the ownership of the data directory and its contents to make sure the server has access. For example, if the data directory is /usr/local/mysql/var, use this command:

  shell> ls -la /usr/local/mysql/var

  If the data directory or its files or subdirectories are not owned by the login account that you use for running the server, change their ownership to that account. If the account is named mysql, use these commands:

  shell> chown -R mysql /usr/local/mysql/var
  shell> chgrp -R mysql /usr/local/mysql/var

  Even with correct ownership, MySQL might fail to start up if there is other security software running on your system that manages application access to various parts of the file system. In this case, reconfigure that software to enable mysqld to access the directories it uses during normal operation.
Testing the Server

• Verify that the network interfaces the server wants to use are available.

If either of the following errors occur, it means that some other program (perhaps another `mysqld` server) is using the TCP/IP port or Unix socket file that `mysqld` is trying to use:

```
Can't start server: Bind on TCP/IP port: Address already in use
Can't start server: Bind on unix socket...
```

Use `ps` to determine whether you have another `mysqld` server running. If so, shut down the server before starting `mysqld` again. (If another server is running, and you really want to run multiple servers, you can find information about how to do so in Running Multiple MySQL Instances on One Machine.)

If no other server is running, execute the command `telnet your_host_name tcp_ip_port_number`. (The default MySQL port number is 3306.) Then press Enter a couple of times. If you do not get an error message like `telnet: Unable to connect to remote host: Connection refused`, some other program is using the TCP/IP port that `mysqld` is trying to use. Track down what program this is and disable it, or tell `mysqld` to listen to a different port with the `--port` option. In this case, specify the same non-default port number for client programs when connecting to the server using TCP/IP.

Another reason the port might be inaccessible is that you have a firewall running that blocks connections to it. If so, modify the firewall settings to permit access to the port.

If the server starts but you cannot connect to it, make sure that you have an entry in `/etc/hosts` that looks like this:

```
127.0.0.1       localhost
```

• If you cannot get `mysqld` to start, try to make a trace file to find the problem by using the `--debug` option. See The DBUG Package.

3.3 Testing the Server

After the data directory is initialized and you have started the server, perform some simple tests to make sure that it works satisfactorily. This section assumes that your current location is the MySQL installation directory and that it has a `bin` subdirectory containing the MySQL programs used here. If that is not true, adjust the command path names accordingly.

Alternatively, add the `bin` directory to your `PATH` environment variable setting. That enables your shell (command interpreter) to find MySQL programs properly, so that you can run a program by typing only its name, not its path name. See Setting Environment Variables.

Use `mysqladmin` to verify that the server is running. The following commands provide simple tests to check whether the server is up and responding to connections:

```
shell> bin/mysqladmin version
shell> bin/mysqladmin variables
```

If you cannot connect to the server, specify a `-u root` option to connect as `root`. If you have assigned a password for the `root` account already, you'll also need to specify `-p` on the command line and enter the password when prompted. For example:

```
shell> bin/mysqladmin -u root -p version
Enter password: (enter root password here)
```

The output from `mysqladmin version` varies slightly depending on your platform and version of MySQL, but should be similar to that shown here:

```
shell> bin/mysqladmin version
mysqladmin  Ver 14.12 Distrib 8.0.22, for pc-linux-gnu on i686 ...
Server version  8.0.22
```
Testing the Server

<table>
<thead>
<tr>
<th>Protocol version</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection</td>
<td>Localhost via UNIX socket</td>
</tr>
<tr>
<td>UNIX socket</td>
<td>/var/lib/mysql/mysql.sock</td>
</tr>
<tr>
<td>Uptime</td>
<td>14 days 5 hours 5 min 21 sec</td>
</tr>
<tr>
<td>Threads</td>
<td>1</td>
</tr>
<tr>
<td>Questions</td>
<td>366</td>
</tr>
<tr>
<td>Slow queries</td>
<td>0</td>
</tr>
<tr>
<td>Opens</td>
<td>0</td>
</tr>
<tr>
<td>Flush tables</td>
<td>1</td>
</tr>
<tr>
<td>Open tables</td>
<td>19</td>
</tr>
<tr>
<td>Queries per second avg</td>
<td>0.000</td>
</tr>
</tbody>
</table>

To see what else you can do with `mysqladmin`, invoke it with the `--help` option.

Verify that you can shut down the server (include a `-p` option if the `root` account has a password already):

```
shell> bin/mysqladmin -u root shutdown
```

Verify that you can start the server again. Do this by using `mysqld_safe` or by invoking `mysqld` directly. For example:

```
shell> bin/mysqld_safe --user=mysql &
```

If `mysqld_safe` fails, see Section 3.2.1, “Troubleshooting Problems Starting the MySQL Server”.

Run some simple tests to verify that you can retrieve information from the server. The output should be similar to that shown here.

Use `mysqlshow` to see what databases exist:

```
shell> bin/mysqlshow
```

The list of installed databases may vary, but always includes at least `mysql` and `information_schema`.

If you specify a database name, `mysqlshow` displays a list of the tables within the database:

```
shell> bin/mysqlshow mysql
Database: mysql
```

---

24
Use the `mysql` program to select information from a table in the `mysql` schema:

```
shell> bin/mysql -e "SELECT User, Host, plugin FROM mysql.user" mysql
```

<table>
<thead>
<tr>
<th>User</th>
<th>Host</th>
<th>plugin</th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>localhost</td>
<td>caching_sha2_password</td>
</tr>
</tbody>
</table>

At this point, your server is running and you can access it. To tighten security if you have not yet assigned a password to the initial account, follow the instructions in Section 3.4, “Securing the Initial MySQL Account”.

For more information about `mysql`, `mysqladmin`, and `mysqlshow`, see `mysql — The MySQL Command-Line Client`, `mysqladmin — Client for Administering a MySQL Server`, and `mysqlshow — Display Database, Table, and Column Information`.

### 3.4 Securing the Initial MySQL Account

The MySQL installation process involves initializing the data directory, including the grant tables in the `mysql` system schema that define MySQL accounts. For details, see Section 3.1, “Initializing the Data Directory”.

This section describes how to assign a password to the initial `root` account created during the MySQL installation procedure, if you have not already done so.

**Note**

Alternative means for performing the process described in this section:

- On Windows, you can perform the process during installation with MySQL Installer (see MySQL Installer for Windows).
- On all platforms, the MySQL distribution includes `mysql_secure_installation`, a command-line utility that automates much of the process of securing a MySQL installation.
- On all platforms, MySQL Workbench is available and offers the ability to manage user accounts (see MySQL Workbench).

A password may already be assigned to the initial account under these circumstances:

- On Windows, installations performed using MySQL Installer give you the option of assigning a password.
- Installation using the macOS installer generates an initial random password, which the installer displays to the user in a dialog box.
- Installation using RPM packages generates an initial random password, which is written to the server error log.
Securing the Initial MySQL Account

- Installations using Debian packages give you the option of assigning a password.

- For data directory initialization performed manually using `mysqld --initialize`, `mysqld` generates an initial random password, marks it expired, and writes it to the server error log. See Section 3.1, “Initializing the Data Directory”.

The `mysql.user` grant table defines the initial MySQL user account and its access privileges. Installation of MySQL creates only a `'root'@'localhost'` superuser account that has all privileges and can do anything. If the `root` account has an empty password, your MySQL installation is unprotected: Anyone can connect to the MySQL server as `root` without a password and be granted all privileges.

The `'root'@'localhost'` account also has a row in the `mysql.proxies_priv` table that enables granting the PROXY privilege for `'@'`, that is, for all users and all hosts. This enables `root` to set up proxy users, as well as to delegate to other accounts the authority to set up proxy users. See Section 4.18, “Proxy Users”.

To assign a password for the initial MySQL `root` account, use the following procedure. Replace `root-password` in the examples with the password that you want to use.

1. If you do not know the initial random password, look in the server error log.
2. Connect to the server as `root` using the password:

```
shell> mysql -u root -p
Enter password: (enter the random root password here)
```
3. Choose a new password to replace the random password:

```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'root-password';
```

- If the `root` account exists but has no password, connect to the server as `root` using no password, then assign a password. This is the case if you initialized the data directory using `mysqld --initialize-insecure`.
1. Connect to the server as `root` using no password:

```
shell> mysql -u root --skip-password
```
2. Assign a password:

```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'root-password';
```

After assigning the `root` account a password, you must supply that password whenever you connect to the server using the account. For example, to connect to the server using the `mysql` client, use this command:

```
shell> mysql -u root -p
Enter password: (enter root password here)
```

To shut down the server with `mysqladmin`, use this command:
3.5 Starting and Stopping MySQL Automatically

This section discusses methods for starting and stopping the MySQL server.

Generally, you start the mysqld server in one of these ways:

- Invoke mysqld directly. This works on any platform.
- On Windows, you can set up a MySQL service that runs automatically when Windows starts. See Starting MySQL as a Windows Service.
- On Unix and Unix-like systems, you can invoke mysqld_safe, which tries to determine the proper options for mysqld and then runs it with those options. See mysqld_safe — MySQL Server Startup Script.
- On Linux systems that support systemd, you can use it to control the server. See Managing MySQL Server with systemd.
- On systems that use System V-style run directories (that is, /etc/init.d and run-level specific directories), invoke mysql.server. This script is used primarily at system startup and shutdown. It usually is installed under the name mysql. The mysql.server script starts the server by invoking mysqld_safe. See mysql.server — MySQL Server Startup Script.
- On macOS, install a launchd daemon to enable automatic MySQL startup at system startup. The daemon starts the server by invoking mysqld_safe. For details, see Installing and Using the MySQL Launch Daemon. A MySQL Preference Pane also provides control for starting and stopping MySQL through the System Preferences. See Installing and Using the MySQL Preference Pane.
- On Solaris, use the service management framework (SMF) system to initiate and control MySQL startup.

systemd, the mysqld_safe and mysql.server scripts, Solaris SMF, and the macOS Startup Item (or MySQL Preference Pane) can be used to start the server manually, or automatically at system startup time. systemd, mysql.server, and the Startup Item also can be used to stop the server.

The following table shows which option groups the server and startup scripts read from option files.

Table 3.1 MySQL Startup Scripts and Supported Server Option Groups

<table>
<thead>
<tr>
<th>Script</th>
<th>Option Groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>mysqld</td>
<td>[mysqld], [server], [mysqld-major_version]</td>
</tr>
<tr>
<td>mysqld_safe</td>
<td>[mysqld], [server], [mysqld_safe]</td>
</tr>
<tr>
<td>mysql.server</td>
<td>[mysqld], [mysql.server], [server]</td>
</tr>
</tbody>
</table>

[mysqld-major_version] means that groups with names like [mysqld-5.7] and [mysqld-8.0] are read by servers having versions 5.7.x, 8.0.x, and so forth. This feature can be used to specify options that can be read only by servers within a given release series.
For backward compatibility, `mysql.server` also reads the `[mysql_server]` group and `mysqld_safe` also reads the `[safe_mysqld]` group. To be current, you should update your option files to use the `[mysql.server]` and `[mysqld_safe]` groups instead.

For more information on MySQL configuration files and their structure and contents, see Using Option Files.
Chapter 4 Access Control and Account Management

Table of Contents

4.1 Account User Names and Passwords .......................................................... 30
4.2 Privileges Provided by MySQL .................................................................. 32
4.3 Grant Tables ......................................................................................... 48
4.4 Specifying Account Names ...................................................................... 56
4.5 Specifying Role Names ........................................................................... 58
4.6 Access Control, Stage 1: Connection Verification .................................... 59
4.7 Access Control, Stage 2: Request Verification .......................................... 62
4.8 Adding Accounts, Assigning Privileges, and Dropping Accounts ............. 63
4.9 Reserved Accounts ............................................................................... 66
4.10 Using Roles ............................................................................................ 67
4.11 Account Categories ............................................................................... 74
4.12 Privilege Restriction Using Partial Revokes ............................................ 77
4.13 When Privilege Changes Take Effect ..................................................... 83
4.14 Assigning Account Passwords ............................................................... 84
4.15 Password Management .......................................................................... 85
4.16 Server Handling of Expired Passwords .................................................. 96
4.17 Pluggable Authentication ....................................................................... 97
4.18 Proxy Users ........................................................................................ 102
4.19 Account Locking ................................................................................ 110
4.20 Setting Account Resource Limits ............................................................ 110
4.21 Troubleshooting Problems Connecting to MySQL .................................... 112
4.22 SQL-Based Account Activity Auditing .................................................... 116

MySQL enables the creation of accounts that permit client users to connect to the server and access data managed by the server. The primary function of the MySQL privilege system is to authenticate a user who connects from a given host and to associate that user with privileges on a database such as SELECT, INSERT, UPDATE, and DELETE. Additional functionality includes the ability to grant privileges for administrative operations.

To control which users can connect, each account can be assigned authentication credentials such as a password. The user interface to MySQL accounts consists of SQL statements such as CREATE USER, GRANT, and REVOKE. See Account Management Statements.

The MySQL privilege system ensures that all users may perform only the operations permitted to them. As a user, when you connect to a MySQL server, your identity is determined by the host from which you connect and the user name you specify. When you issue requests after connecting, the system grants privileges according to your identity and what you want to do.

MySQL considers both your host name and user name in identifying you because there is no reason to assume that a given user name belongs to the same person on all hosts. For example, the user joe who connects from office.example.com need not be the same person as the user joe who connects from home.example.com. MySQL handles this by enabling you to distinguish users on different hosts that happen to have the same name: You can grant one set of privileges for connections by joe from office.example.com, and a different set of privileges for connections by joe from home.example.com. To see what privileges a given account has, use the SHOW GRANTS statement. For example:

```
SHOW GRANTS FOR 'joe'@'office.example.com';
SHOW GRANTS FOR 'joe'@'home.example.com';
```

Internally, the server stores privilege information in the grant tables of the mysql system database. The MySQL server reads the contents of these tables into memory when it starts and bases access-control decisions on the in-memory copies of the grant tables.
MySQL access control involves two stages when you run a client program that connects to the server:

**Stage 1:** The server accepts or rejects the connection based on your identity and whether you can verify your identity by supplying the correct password.

**Stage 2:** Assuming that you can connect, the server checks each statement you issue to determine whether you have sufficient privileges to perform it. For example, if you try to select rows from a table in a database or drop a table from the database, the server verifies that you have the `SELECT` privilege for the table or the `DROP` privilege for the database.

For a more detailed description of what happens during each stage, see Section 4.6, “Access Control, Stage 1: Connection Verification”, and Section 4.7, “Access Control, Stage 2: Request Verification”. For help in diagnosing privilege-related problems, see Section 4.21, “Troubleshooting Problems Connecting to MySQL”.

If your privileges are changed (either by yourself or someone else) while you are connected, those changes do not necessarily take effect immediately for the next statement that you issue. For details about the conditions under which the server reloads the grant tables, see Section 4.13, “When Privilege Changes Take Effect”.

There are some things that you cannot do with the MySQL privilege system:

- You cannot explicitly specify that a given user should be denied access. That is, you cannot explicitly match a user and then refuse the connection.
- You cannot specify that a user has privileges to create or drop tables in a database but not to create or drop the database itself.
- A password applies globally to an account. You cannot associate a password with a specific object such as a database, table, or routine.

### 4.1 Account User Names and Passwords

MySQL stores accounts in the `user` table of the `mysql` system database. An account is defined in terms of a user name and the client host or hosts from which the user can connect to the server. For information about account representation in the `user` table, see Section 4.3, “Grant Tables”.

An account may also have authentication credentials such as a password. The credentials are handled by the account authentication plugin. MySQL supports multiple authentication plugins. Some of them use built-in authentication methods, whereas others enable authentication using external authentication methods. See Section 4.17, “Pluggable Authentication”.

There are several distinctions between the way user names and passwords are used by MySQL and your operating system:

- User names, as used by MySQL for authentication purposes, have nothing to do with user names (login names) as used by Windows or Unix. On Unix, most MySQL clients by default try to log in using the current Unix user name as the MySQL user name, but that is for convenience only. The default can be overridden easily, because client programs permit any user name to be specified with a `-u` or `--user` option. This means that anyone can attempt to connect to the server using any user name, so you cannot make a database secure in any way unless all MySQL accounts have passwords. Anyone who specifies a user name for an account that has no password can connect successfully to the server.
- MySQL user names are up to 32 characters long. Operating system user names may have a different maximum length.

**Warning**

The MySQL user name length limit is hardcoded in MySQL servers and clients, and trying to circumvent it by modifying the definitions of the tables in the `mysql` database does not work.
You should never alter the structure of tables in the mysql database in any manner whatsoever except by means of the procedure that is described in Upgrading MySQL. Attempting to redefine MySQL’s system tables in any other fashion results in undefined and unsupported behavior. The server is free to ignore rows that become malformed as a result of such modifications.

- To authenticate client connections for accounts that use built-in authentication methods, the server uses passwords stored in the user table. These passwords are distinct from passwords for logging in to your operating system. There is no necessary connection between the “external” password you use to log in to a Windows or Unix machine and the password you use to access the MySQL server on that machine.

If the server authenticates a client using some other plugin, the authentication method that the plugin implements may or may not use a password stored in the user table. In this case, it is possible that an external password is also used to authenticate to the MySQL server.

- Passwords stored in the user table are encrypted using plugin-specific algorithms.

- If the user name and password contain only ASCII characters, it is possible to connect to the server regardless of character set settings. To enable connections when the user name or password contain non-ASCII characters, client applications should call the mysql_options() C API function with the MYSQL_SET_CHARSET_NAME option and appropriate character set name as arguments. This causes authentication to take place using the specified character set. Otherwise, authentication fails unless the server default character set is the same as the encoding in the authentication defaults.

Standard MySQL client programs support a --default-character-set option that causes mysql_options() to be called as just described. In addition, character set autodetection is supported as described in Connection Character Sets and Collations. For programs that use a connector that is not based on the C API, the connector may provide an equivalent to mysql_options() that can be used instead. Check the connector documentation.

The preceding notes do not apply for ucs2, utf16, and utf32, which are not permitted as client character sets.

The MySQL installation process populates the grant tables with an initial root account, as described in Section 3.4, “Securing the Initial MySQL Account”, which also discusses how to assign a password to it. Thereafter, you normally set up, modify, and remove MySQL accounts using statements such as CREATE USER, DROP USER, GRANT, and REVOKE. See Section 4.8, “Adding Accounts, Assigning Privileges, and Dropping Accounts”, and Account Management Statements.

To connect to a MySQL server with a command-line client, specify user name and password options as necessary for the account that you want to use:

```
shell> mysql --user=finley --password db_name
```

If you prefer short options, the command looks like this:

```
shell> mysql -u finley -p db_name
```

If you omit the password value following the --password or -p option on the command line (as just shown), the client prompts for one. Alternatively, the password can be specified on the command line:

```
shell> mysql --user=finley --password=password db_name
shell> mysql -u finley -ppassword db_name
```

If you use the -p option, there must be no space between -p and the following password value.

Specifying a password on the command line should be considered insecure. See Section 2.2.1, “End-User Guidelines for Password Security”. To avoid giving the password on the command line, use an option file or a login path file. See Using Option Files, and mysql_config_editor — MySQL Configuration Utility.
For additional information about specifying user names, passwords, and other connection parameters, see *Connecting to the MySQL Server Using Command Options*.

### 4.2 Privileges Provided by MySQL

The privileges granted to a MySQL account determine which operations the account can perform. MySQL privileges differ in the contexts in which they apply and at different levels of operation:

- Administrative privileges enable users to manage operation of the MySQL server. These privileges are global because they are not specific to a particular database.
- Database privileges apply to a database and to all objects within it. These privileges can be granted for specific databases, or globally so that they apply to all databases.
- Privileges for database objects such as tables, indexes, views, and stored routines can be granted for specific objects within a database, for all objects of a given type within a database (for example, all tables in a database), or globally for all objects of a given type in all databases.

Privileges also differ in terms of whether they are static (built in to the server) or dynamic (defined at runtime). Whether a privilege is static or dynamic affects its availability to be granted to user accounts and roles. For information about the differences between static and dynamic privileges, see *Static Versus Dynamic Privileges*.

Information about account privileges is stored in the grant tables in the *mysql* system database. For a description of the structure and contents of these tables, see Section 4.3, "Grant Tables". The MySQL server reads the contents of the grant tables into memory when it starts, and reloads them under the circumstances indicated in Section 4.13, "When Privilege Changes Take Effect". The server bases access-control decisions on the in-memory copies of the grant tables.

**Important**

Some MySQL releases introduce changes to the grant tables to add new privileges or features. To make sure that you can take advantage of any new capabilities, update your grant tables to the current structure whenever you upgrade MySQL. See *Upgrading MySQL*.

The following sections summarize the available privileges, provide more detailed descriptions of each privilege, and offer usage guidelines.

- **Summary of Available Privileges**
- **Static Privilege Descriptions**
- **Dynamic Privilege Descriptions**
- **Privilege-Granting Guidelines**
- **Static Versus Dynamic Privileges**
- **Migrating Accounts from SUPER to Dynamic Privileges**

#### Summary of Available Privileges

The following table shows the static privilege names used in *GRANT* and *REVOKE* statements, along with the column name associated with each privilege in the grant tables and the context in which the privilege applies.

**Table 4.1 Permissible Static Privileges for GRANT and REVOKE**

<table>
<thead>
<tr>
<th>Privilege</th>
<th>Grant Table Column</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL [PRIVILEGES]</td>
<td>Synonym for “all privileges“</td>
<td>Server administration</td>
</tr>
<tr>
<td>ALTER</td>
<td>Alter_priv</td>
<td>Tables</td>
</tr>
</tbody>
</table>
Summary of Available Privileges

<table>
<thead>
<tr>
<th>Privilege</th>
<th>Grant Table Column</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER ROUTINE</td>
<td>Alter_routine_priv</td>
<td>Stored routines</td>
</tr>
<tr>
<td>CREATE</td>
<td>Create_priv</td>
<td>Databases, tables, or indexes</td>
</tr>
<tr>
<td>CREATE ROLE</td>
<td>Create_role_priv</td>
<td>Server administration</td>
</tr>
<tr>
<td>CREATE ROUTINE</td>
<td>Create_routine_priv</td>
<td>Stored routines</td>
</tr>
<tr>
<td>CREATE TABLESPACE</td>
<td>Create_tablespace_priv</td>
<td>Server administration</td>
</tr>
<tr>
<td>CREATE TEMPORARY TABLES</td>
<td>Create_tmp_table_priv</td>
<td>Tables</td>
</tr>
<tr>
<td>CREATE USER</td>
<td>Create_user_priv</td>
<td>Server administration</td>
</tr>
<tr>
<td>CREATE VIEW</td>
<td>Create_view_priv</td>
<td>Views</td>
</tr>
<tr>
<td>DELETE</td>
<td>Delete_priv</td>
<td>Tables</td>
</tr>
<tr>
<td>DROP</td>
<td>Drop_priv</td>
<td>Databases, tables, or views</td>
</tr>
<tr>
<td>DROP ROLE</td>
<td>Drop_role_priv</td>
<td>Server administration</td>
</tr>
<tr>
<td>EVENT</td>
<td>Event_priv</td>
<td>Databases</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>Execute_priv</td>
<td>Stored routines</td>
</tr>
<tr>
<td>FILE</td>
<td>File_priv</td>
<td>File access on server host</td>
</tr>
<tr>
<td>GRANT OPTION</td>
<td>Grant_priv</td>
<td>Databases, tables, or stored routines</td>
</tr>
<tr>
<td>INDEX</td>
<td>Index_priv</td>
<td>Tables</td>
</tr>
<tr>
<td>INSERT</td>
<td>Insert_priv</td>
<td>Tables or columns</td>
</tr>
<tr>
<td>LOCK TABLES</td>
<td>Lock_tables_priv</td>
<td>Databases</td>
</tr>
<tr>
<td>PROCESS</td>
<td>Process_priv</td>
<td>Server administration</td>
</tr>
<tr>
<td>PROXY</td>
<td>See proxies_priv table</td>
<td>Server administration</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>References_priv</td>
<td>Databases or tables</td>
</tr>
<tr>
<td>RELOAD</td>
<td>Reload_priv</td>
<td>Server administration</td>
</tr>
<tr>
<td>REPLICATION CLIENT</td>
<td>Repl_client_priv</td>
<td>Server administration</td>
</tr>
<tr>
<td>REPLICATION SLAVE</td>
<td>Repl_slave_priv</td>
<td>Server administration</td>
</tr>
<tr>
<td>SELECT</td>
<td>Select_priv</td>
<td>Tables or columns</td>
</tr>
<tr>
<td>SHOW DATABASES</td>
<td>Show_db_priv</td>
<td>Server administration</td>
</tr>
<tr>
<td>SHOW VIEW</td>
<td>Show_view_priv</td>
<td>Views</td>
</tr>
<tr>
<td>SHUTDOWN</td>
<td>Shutdown_priv</td>
<td>Server administration</td>
</tr>
<tr>
<td>SUPER</td>
<td>Super_priv</td>
<td>Server administration</td>
</tr>
<tr>
<td>TRIGGER</td>
<td>Trigger_priv</td>
<td>Tables</td>
</tr>
<tr>
<td>UPDATE</td>
<td>Update_priv</td>
<td>Tables or columns</td>
</tr>
<tr>
<td>USAGE</td>
<td>Synonym for “no privileges”</td>
<td>Server administration</td>
</tr>
</tbody>
</table>

The following table shows the dynamic privilege names used in \texttt{GRANT} and \texttt{REVOKE} statements, along with the context in which the privilege applies.

\textbf{Table 4.2 Permissible Dynamic Privileges for GRANT and REVOKE}

<table>
<thead>
<tr>
<th>Privilege</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLICATION_PASSWORD_ADMIN</td>
<td>Dual password administration</td>
</tr>
<tr>
<td>AUDIT_ADMIN</td>
<td>Audit log administration</td>
</tr>
<tr>
<td>BACKUP_ADMIN</td>
<td>Backup administration</td>
</tr>
<tr>
<td>BINLOG_ADMIN</td>
<td>Backup and Replication administration</td>
</tr>
</tbody>
</table>
## Static Privilege Descriptions

Static privileges are built into the server, in contrast to dynamic privileges, which are defined at runtime. The following list describes each static privilege available in MySQL.

Particular SQL statements might have more specific privilege requirements than indicated here. If so, the description for the statement in question provides the details.

- **ALL, ALL PRIVILEGES**
  These privilege specifiers are shorthand for “all privileges available at a given privilege level” (except GRANT OPTION). For example, granting **ALL** at the global or table level grants all global privileges or all table-level privileges, respectively.

- **ALTER**
  Enables use of the **ALTER TABLE** statement to change the structure of tables. **ALTER TABLE** also requires the **CREATE** and **INSERT** privileges. Renaming a table requires **ALTER** and **DROP** on the old table, **CREATE**, and **INSERT** on the new table.

- **ALTER ROUTINE**
  Enables use of statements that alter or drop stored routines (stored procedures and functions). For routines that fall within the scope at which the privilege is granted and for which the user is not the
user named as the routine **DEFINER**, also enables access to routine properties other than the routine definition.

- **CREATE**

  Enables use of statements that create new databases and tables.

- **CREATE ROLE**

  Enables use of the **CREATE ROLE** statement. (The **CREATE USER** privilege also enables use of the **CREATE ROLE** statement.) See Section 4.10, “Using Roles”.

  The **CREATE ROLE** and **DROP ROLE** privileges are not as powerful as **CREATE USER** because they can be used only to create and drop accounts. They cannot be used as **CREATE USER** can be modify account attributes or rename accounts. See User and Role Interchangeability.

- **CREATE ROUTINE**

  Enables use of statements that create stored routines (stored procedures and functions). For routines that fall within the scope at which the privilege is granted and for which the user is not the user named as the routine **DEFINER**, also enables access to routine properties other than the routine definition.

- **CREATE TABLESPACE**

  Enables use of statements that create, alter, or drop tablespaces and log file groups.

- **CREATE TEMPORARY TABLES**

  Enables the creation of temporary tables using the **CREATE TEMPORARY TABLE** statement.

  After a session has created a temporary table, the server performs no further privilege checks on the table. The creating session can perform any operation on the table, such as **DROP TABLE**, **INSERT**, **UPDATE**, or **SELECT**. For more information, see CREATE TEMPORARY TABLE Statement.

- **CREATE USER**


- **CREATE VIEW**

  Enables use of the **CREATE VIEW** statement.

- **DELETE**

  Enables rows to be deleted from tables in a database.

- **DROP**

  Enables use of statements that drop (remove) existing databases, tables, and views. The **DROP** privilege is required to use the **ALTER TABLE ... DROP PARTITION** statement on a partitioned table. The **DROP** privilege is also required for **TRUNCATE TABLE**.

- **DROP ROLE**

  Enables use of the **DROP ROLE** statement. (The **CREATE USER** privilege also enables use of the **DROP ROLE** statement.) See Section 4.10, “Using Roles”.

  The **CREATE ROLE** and **DROP ROLE** privileges are not as powerful as **CREATE USER** because they can be used only to create and drop accounts. They cannot be used as **CREATE USER** can be modify account attributes or rename accounts. See User and Role Interchangeability.
Static Privilege Descriptions

• **EVENT**

  Enables use of statements that create, alter, drop, or display events for the Event Scheduler.

• **EXECUTE**

  Enables use of statements that execute stored routines (stored procedures and functions). For routines that fall within the scope at which the privilege is granted and for which the user is not the user named as the routine **DEFINER**, also enables access to routine properties other than the routine definition.

• **FILE**

  Affects the following operations and server behaviors:

  • Enables reading and writing files on the server host using the **LOAD DATA** and **SELECT ... INTO OUTFILE** statements and the **LOAD_FILE()** function. A user who has the **FILE** privilege can read any file on the server host that is either world-readable or readable by the MySQL server. (This implies the user can read any file in any database directory, because the server can access any of those files.)

  • Enables creating new files in any directory where the MySQL server has write access. This includes the server's data directory containing the files that implement the privilege tables.

  • Enables use of the **DATA DIRECTORY** or **INDEX DIRECTORY** table option for the **CREATE TABLE** statement.

  As a security measure, the server does not overwrite existing files.

  To limit the location in which files can be read and written, set the **secure_file_priv** system variable to a specific directory. See **Server System Variables**.

• **GRANT OPTION**

  Enables you to grant to or revoke from other users those privileges that you yourself possess.

• **INDEX**

  Enables use of statements that create or drop (remove) indexes. **INDEX** applies to existing tables. If you have the **CREATE** privilege for a table, you can include index definitions in the **CREATE TABLE** statement.

• **INSERT**

  Enables rows to be inserted into tables in a database. **INSERT** is also required for the **ANALYZE TABLE**, **OPTIMIZE TABLE**, and **REPAIR TABLE** table-maintenance statements.

• **LOCK TABLES**

  Enables use of explicit **LOCK TABLES** statements to lock tables for which you have the **SELECT** privilege. This includes use of write locks, which prevents other sessions from reading the locked table.

• **PROCESS**

  Enables display of information about the threads executing within the server (that is, information about the statements being executed by sessions). The privilege enables use of **SHOW PROCESSLIST** or **mysqladmin processlist** to see threads belonging to other accounts; you can always see your own threads. The **PROCESS** privilege also enables use of **SHOW ENGINE**.
Static Privilege Descriptions

- **PROXY**
  Enables one user to impersonate or become known as another user. See Section 4.18, “Proxy Users”.

- **REFERENCES**
  Creation of a foreign key constraint requires the REFERENCES privilege for the parent table.

- **RELOAD**
  Enables use of the FLUSH statement. It also enables mysqladmin commands that are equivalent to FLUSH operations: flush-hosts, flush-logs, flush-privileges, flush-status, flush-tables, flush-threads, refresh, and reload.

  The reload command tells the server to reload the grant tables into memory. flush-privileges is a synonym for reload. The refresh command closes and reopens the log files and flushes all tables. The other flush-xxx commands perform functions similar to refresh, but are more specific and may be preferable in some instances. For example, if you want to flush just the log files, flush-logs is a better choice than refresh.

  The RELOAD privilege also enables use of the RESET MASTER and RESET SLAVE statements.

- **REPLICATION CLIENT**
  Enables use of the SHOW MASTER STATUS, SHOW SLAVE STATUS, and SHOW BINARY LOGS statements. Grant this privilege to accounts that are used by slave servers to connect to the current server as their master.

- **REPLICATION SLAVE**
  Enables the account to request updates that have been made to databases on the master server, using the SHOW SLAVE HOSTS, SHOW RELAYLOG EVENTS, and SHOW BINLOG EVENTS statements. This privilege is also required to use the mysqlbinlog options --read-from-remote-server (-R) and --read-from-remote-master. Grant this privilege to accounts that are used by slave servers to connect to the current server as their master.

- **SELECT**
  Enables rows to be selected from tables in a database. SELECT statements require the SELECT privilege only if they actually access tables. Some SELECT statements do not access tables and can be executed without permission for any database. For example, you can use SELECT as a simple calculator to evaluate expressions that make no reference to tables:

  ```
  SELECT 1+1;
  SELECT PI()*2;
  ```

  The SELECT privilege is also needed for other statements that read column values. For example, SELECT is needed for columns referenced on the right hand side of col_name=expr assignment in UPDATE statements or for columns named in the WHERE clause of DELETE or UPDATE statements.

  The SELECT privilege is needed for tables or views used with EXPLAIN, including any underlying tables in view definitions.
• **SHOW DATABASES**

Enables the account to see database names by issuing the `SHOW DATABASE` statement. Accounts that do not have this privilege see only databases for which they have some privileges, and cannot use the statement at all if the server was started with the `--skip-show-database` option.

**Caution**

Because any static global privilege is considered a privilege for all databases, any static global privilege enables a user to see all database names with `SHOW DATABASES` or by examining the `SCHEMATA` table of `INFORMATION_SCHEMA`, except databases that have been restricted at the database level by partial revokes.

• **SHOW VIEW**

Enables use of the `SHOW CREATE VIEW` statement. This privilege is also needed for views used with `EXPLAIN`.

• **SHUTDOWN**

Enables use of the `SHUTDOWN` and `RESTART` statements, the `mysqladmin shutdown` command, and the `mysql_shutdown()` C API function.

• **SUPER**

`SUPER` is a powerful and far-reaching privilege and should not be granted lightly. If an account needs to perform only a subset of `SUPER` operations, it may be possible to achieve the desired privilege set.
by instead granting one or more dynamic privileges, each of which confers more limited capabilities. See Dynamic Privilege Descriptions.

Note

SUPER is deprecated and will be removed in a future version of MySQL. See Migrating Accounts from SUPER to Dynamic Privileges.

SUPER affects the following operations and server behaviors:

- Enables system variable changes at runtime:
  - Enables server configuration changes to global system variables with `SET GLOBAL` and `SET PERSIST`.
    The corresponding dynamic privilege is `SYSTEM_VARIABLES_ADMIN`.
  - Enables setting restricted session system variables that require a special privilege.
    The corresponding dynamic privilege is `SESSION_VARIABLES_ADMIN`.

See also System Variable Privileges.

- Enables changes to global transaction characteristics (see SET TRANSACTION Statement).
  The corresponding dynamic privilege is `SYSTEM_VARIABLES_ADMIN`.

- Enables the account to start and stop replication, including Group Replication.
  The corresponding dynamic privilege is `REPLICATION_SLAVE_ADMIN` for regular replication, `GROUP_REPLICATION_ADMIN` for Group Replication.

- Enables use of the `CHANGE MASTER TO` and `CHANGE REPLICATION FILTER` statements.
  The corresponding dynamic privilege is `REPLICATION_SLAVE_ADMIN`.

- Enables binary log control by means of the `PURGE BINARY LOGS` and `BINLOG` statements.
  The corresponding dynamic privilege is `BINLOG_ADMIN`.

- Enables setting the effective authorization ID when executing a view or stored program. A user with this privilege can specify any account in the `DEFINER` attribute of a view or stored program.
  The corresponding dynamic privilege is `SET_USER_ID`.

- Enables use of the `CREATE SERVER`, `ALTER SERVER`, and `DROP SERVER` statements.

- Enables use of the `mysqladmin debug` command.

- Enables InnoDB encryption key rotation.
  The corresponding dynamic privilege is `ENCRYPTION_KEY_ADMIN`.

- Enables execution of Version Tokens user-defined functions.
  The corresponding dynamic privilege is `VERSION_TOKEN_ADMIN`.

- Enables granting and revoking roles, use of the `WITH ADMIN OPTION` clause of the `GRANT` statement, and nonempty `<graphml>` element content in the result from the `ROLES_GRAPHML()` function.
  The corresponding dynamic privilege is `ROLE_ADMIN`. 
Dynamic Privilege Descriptions

- Enables control over client connections not permitted to non-SUPER accounts:
  - Enables use of the KILL statement or mysqladmin kill command to kill threads belonging to other accounts. (An account can always kill its own threads.)
- The server does not execute init_connect system variable content when SUPER clients connect.
- The server accepts one connection from a SUPER client even if the connection limit configured by the max_connections system variable is reached.
- A server in offline mode (offline_mode enabled) does not terminate SUPER client connections at the next client request, and accepts new connections from SUPER clients.
- Updates can be performed even when the read_only system variable is enabled. This applies to explicit table updates, and to use of account-management statements such as GRANT and REVOKE that update tables implicitly.
  
  The corresponding dynamic privilege for the preceding connection-control operations is CONNECTION_ADMIN.

You may also need the SUPER privilege to create or alter stored functions if binary logging is enabled, as described in Stored Program Binary Logging.

- TRIGGER

  Enables trigger operations. You must have this privilege for a table to create, drop, execute, or display triggers for that table.

  When a trigger is activated (by a user who has privileges to execute INSERT, UPDATE, or DELETE statements for the table associated with the trigger), trigger execution requires that the user who defined the trigger still have the TRIGGER privilege for the table.

- UPDATE

  Enables rows to be updated in tables in a database.

- USAGE

  This privilege specifier stands for “no privileges.” It is used at the global level with GRANT to specify clauses such as WITH GRANT OPTION without naming specific account privileges in the privilege list. SHOW GRANTS displays USAGE to indicate that an account has no privileges at a privilege level.

Dynamic Privilege Descriptions

Dynamic privileges are defined at runtime, in contrast to static privileges, which are built in to the server. The following list describes each dynamic privilege available in MySQL.

Most dynamic privileges are defined at server startup. Others are defined by a particular server component or plugin, as indicated in the privilege descriptions. In such cases, the privilege is unavailable unless the component or plugin that defines it is enabled.

Particular SQL statements might have more specific privilege requirements than indicated here. If so, the description for the statement in question provides the details.

- APPLICATION_PASSWORD_ADMIN (added in MySQL 8.0.14)

  For dual-password capability, this privilege enables use of the RETAIN CURRENT PASSWORD and DISCARD OLD PASSWORD clauses for ALTER USER and SET PASSWORD statements that apply to your own account. This privilege is required to manipulate your own secondary password because most users require only one password.
If an account is to be permitted to manipulate secondary passwords for all accounts, it should be granted the `CREATE USER` privilege rather than `APPLICATION_PASSWORD_ADMIN`.

For more information about use of dual passwords, see Section 4.15, “Password Management”.

- **AUDIT_ADMIN**
  Enables audit log configuration. This privilege is defined by the `audit_log` plugin; see Section 6.5, “MySQL Enterprise Audit”.

- **BACKUP_ADMIN**
  Enables execution of the `LOCK INSTANCE FOR BACKUP` statement and access to the Performance Schema `log_status` table.

  **Note**
  Besides `BACKUP_ADMIN`, the `SELECT` privilege on the `log_status` table is also needed for its access.

  The `BACKUP_ADMIN` privilege is automatically granted to users with the `RELOAD` privilege when performing an in-place upgrade to MySQL 8.0 from an earlier version.

- **BINLOG_ADMIN**
  Enables binary log control by means of the `PURGE BINARY LOGS` and `BINLOG` statements.

- **BINLOG_ENCRYPTION_ADMIN**
  Enables setting the system variable `binlog_encryption`, which activates or deactivates encryption for binary log files and relay log files. This ability is not provided by the `BINLOG_ADMIN`, `SYSTEM_VARIABLES_ADMIN`, or `SESSION_VARIABLES_ADMIN` privileges. The related system variable `binlog_rotate_encryption_master_key_at_startup`, which rotates the binary log master key automatically when the server is restarted, does not require this privilege.

- **CLONE_ADMIN**
  Enables execution of the `CLONE` statements. Includes `BACKUP_ADMIN` and `SHUTDOWN` privileges.

- **CONNECTION_ADMIN**
  Enables use of the `KILL` statement or `mysqladmin kill` command to kill threads belonging to other accounts. (An account can always kill its own threads.)

  Enables setting system variables related to client connections, or circumventing restrictions related to client connections. `CONNECTION_ADMIN` applies to the effects of these system variables:

  - `init_connect`: The server does not execute `init_connect` system variable content when `CONNECTION_ADMIN` clients connect.

  - `max_connections`: The server accepts one connection from a `CONNECTION_ADMIN` client even if the connection limit configured by the `max_connections` system variable is reached.

  - `offline_mode`: A server in offline mode (offline_mode enabled) does not terminate `CONNECTION_ADMIN` client connections at the next client request, and accepts new connections from `CONNECTION_ADMIN` clients.

  - `read_only`: Updates can be performed even when the `read_only` system variable is enabled. This applies to explicit table updates, and to use of account-management statements such as `GRANT` and `REVOKE` that update tables implicitly.

- **ENCRYPTION_KEY_ADMIN**
Dynamic Privilege Descriptions

Enables **InnoDB** encryption key rotation.

- **FIREWALL_ADMIN**
  Enables a user to administer firewall rules for any user. This privilege is defined by the `MYSQL_FIREWALL` plugin; see Section 6.7, “MySQL Enterprise Firewall”.

- **FIREWALL_USER**
  Enables users to update their own firewall rules. This privilege is defined by the `MYSQL_FIREWALL` plugin; see Section 6.7, “MySQL Enterprise Firewall”.

- **GROUP_REPLICATION_ADMIN**
  Enables the account to start and stop Group Replication using the `START GROUP REPLICATION` and `STOP GROUP REPLICATION` statements, to change the global setting for the `group_replication_consistency` system variable, and to use the `group_replication_set_write_concurrency()` and `group_replication_set_communication_protocol()` UDFs. Grant this privilege to accounts that are used to administer servers that are members of a replication group.

- **INNODB_REDO_LOG_ARCHIVE**
  Enables the account to activate and deactivate redo log archiving.

- **NDB_STORED_USER**
  Enables the user or role and its privileges to be shared and synchronized between all **NDB**-enabled MySQL servers as soon as they join a given **NDB** Cluster. This privilege is available only if the **NDB** storage engine is enabled.

  Any changes to or revocations of privileges made for the given user or role are synchronized immediately with all connected MySQL servers (SQL nodes). You should be aware that there is no guarantee that multiple statements affecting privileges originating from different SQL nodes are executed on all SQL nodes in the same order. For this reason, it is highly recommended that all user administration be done from a single designated SQL node.

  **NDB_STORED_USER** is a global privilege and must be granted or revoked using `ON *.*`. Trying to set any other scope for this privilege results in an error. This privilege can be given to most application and administrative users, but it cannot be granted to system reserved accounts such as `mysql.session@localhost` or `mysql.infoschema@localhost`.

  A user that has been granted the **NDB_STORED_USER** privilege is stored in **NDB** (and thus shared by all SQL nodes), as is a role with this privilege. A user that is merely granted a role that has **NDB_STORED_USER** is not stored in **NDB**; each **NDB** stored user must be granted the privilege explicitly.

  For more detailed information about how this works in **NDB**, see Distributed MySQL Privileges with **NDB_STORED_USER**.

  The **NDB_STORED_USER** privilege is available beginning with **NDB** 8.0.18.

- **PERSIST_RO_VARIABLES_ADMIN**
  For users who also have **SYSTEM_VARIABLES_ADMIN**, **PERSIST_RO_VARIABLES_ADMIN** enables use of `SET PERSIST_ONLY` to persist global system variables to the `mysqld-auto.cnf` option file in the data directory. This statement is similar to `SET PERSIST` but does not modify the runtime global system variable value. This makes `SET PERSIST_ONLY` suitable for configuring read-only system variables that can be set only at server startup.

  See also System Variable Privileges.
Dynamic Privilege Descriptions

• **REPLICATION_APPLIER**

  Enables the account to act as the PRIVILEGE_CHECKS_USER for a replication channel, and to execute BINLOG statements in mysqlbinlog output. Grant this privilege to accounts that are assigned using CHANGE MASTER TO to provide a security context for replication channels, and to handle replication errors on those channels. As well as the REPLICATION_APPLIER privilege, you must also give the account the required privileges to execute the transactions received by the replication channel or contained in the mysqlbinlog output, for example to update the affected tables. For more information, see Replication Privilege Checks.

• **REPLICATION_SLAVE_ADMIN**

  Enables the account to connect to the master server, start and stop replication using the START SLAVE and STOP SLAVE statements, and use the CHANGE MASTER TO and CHANGE REPLICATION FILTER statements. Grant this privilege to accounts that are used by slave servers to connect to the current server as their master. This privilege does not apply to Group Replication; use GROUP_REPLICATION_ADMIN for that.

• **RESOURCE_GROUP_ADMIN**

  Enables resource group management, consisting of creating, altering, and dropping resource groups, and assignment of threads and statements to resource groups. A user with this privilege can perform any operation relating to resource groups.

• **RESOURCE_GROUP_USER**

  Enables assigning threads and statements to resource groups. A user with this privilege can use the SET RESOURCE GROUP statement and the RESOURCE_GROUP optimizer hint.

• **ROLE_ADMIN**

  Enables granting and revoking roles, use of the WITH ADMIN OPTION clause of the GRANT statement, and nonempty GRAPHML element content in the result from the ROLES_GRAPHML() function. Required to set the value of the mandatory_roles system variable.

• **SERVICE_CONNECTION_ADMIN**

  Enables connections to the network interface that permits only administrative connections (see How MySQL Handles Client Connections).

• **SESSION_VARIABLES_ADMIN** (added in MySQL 8.0.14)

  For most system variables, setting the session value requires no special privileges and can be done by any user to affect the current session. For some system variables, setting the session value can have effects outside the current session and thus is a restricted operation. For these, the SESSION_VARIABLES_ADMIN privilege enables the user to set the session value.

  If a system variable is restricted and requires a special privilege to set the session value, the variable description indicates that restriction. Examples include binlog_format, sql_log_bin, and sql_log_off.

  Prior to MySQL 8.0.14 when SESSION_VARIABLES_ADMIN was added, restricted session system variables can be set only by users who have the SYSTEM_VARIABLES_ADMIN or SUPER privilege.

  The SESSION_VARIABLES_ADMIN privilege is a subset of the SYSTEM_VARIABLES_ADMIN and SUPER privileges. A user who has either of those privileges is also permitted to set restricted session variables and effectively has SESSION_VARIABLES_ADMIN by implication and need not be granted SESSION_VARIABLES_ADMIN explicitly.

  See also System Variable Privileges.

• **SET_USER_ID**
Dynamic Privilege Descriptions

Enables setting the effective authorization ID when executing a view or stored program. A user with this privilege can specify any account for the DEFINER attribute of a view or stored program.

- **SHOW_ROUTINE** (added in MySQL 8.0.20)

  Enables a user to access definitions and properties of all stored routines (stored procedures and functions), even those for which the user is not named as the routine DEFINER. This access includes:

  - The contents of the INFORMATION_SCHEMA.ROUTINES table.
  - The SHOW CREATE FUNCTION and SHOW CREATE PROCEDURE statements.
  - The SHOW FUNCTION CODE and SHOW PROCEDURE CODE statements.
  - The SHOW FUNCTION STATUS and SHOW PROCEDURE STATUS statements.

Prior to MySQL 8.0.20, for a user to access definitions of routines the user did not define, the user must have the global SELECT privilege, which is very broad. As of 8.0.20, SHOW_ROUTINE may be granted instead as a privilege with a more restricted scope that permits access to routine definitions. (That is, an administrator can rescind global SELECT from users that do not otherwise require it and grant SHOW_ROUTINE instead.) This enables an account to back up stored routines without requiring a broad privilege.

- **SYSTEM_USER** (added in MySQL 8.0.16)

  The SYSTEM_USER privilege distinguishes system users from regular users:

  - A user with the SYSTEM_USER privilege is a system user.
  - A user without the SYSTEM_USER privilege is a regular user.

  The SYSTEM_USER privilege has an effect on the accounts to which a given user can apply its other privileges, as well as whether the user is protected from other accounts:

  - A system user can modify both system and regular accounts. That is, a user who has the appropriate privileges to perform a given operation on regular accounts is enabled by possession of SYSTEM_USER to also perform the operation on system accounts. A system account can be modified only by system users with appropriate privileges, not by regular users.
  - A regular user with appropriate privileges can modify regular accounts, but not system accounts. A regular account can be modified by both system and regular users with appropriate privileges.

  For more information, see Section 4.11, “Account Categories”.

  The protection against modification by regular accounts that is afforded to system accounts by the SYSTEM_USER privilege does not apply to regular accounts that have privileges on the mysql system schema and thus can directly modify the grant tables in that schema. For full protection, do not grant mysql schema privileges to regular accounts. See Protecting System Accounts Against Manipulation by Regular Accounts.
Privilege-Granting Guidelines

- **SYSTEM_VARIABLES_ADMIN**

  Affects the following operations and server behaviors:

  - Enables system variable changes at runtime:
    - Enables server configuration changes to global system variables with `SET GLOBAL` and `SET PERSIST`.
    - Enables server configuration changes to global system variables with `SET PERSIST_ONLY`, if the user also has `PERSIST_RO_VARIABLES_ADMIN`.
    - Enables setting restricted session system variables that require a special privilege. In effect, `SYSTEM_VARIABLES_ADMIN` implies `SESSION_VARIABLES_ADMIN` without explicitly granting `SESSION_VARIABLES_ADMIN`.

  See also System Variable Privileges.

  - Enables changes to global transaction characteristics (see `SET TRANSACTION Statement`).

- **TABLE_ENCRYPTION_ADMIN** (added in MySQL 8.0.16)

  Enables a user to override default encryption settings when `table_encryption_privilege_check` is enabled; see Defining an Encryption Default for Schemas and General Tablespaces.

- **VERSION_TOKEN_ADMIN**

  Enables execution of Version Tokens user-defined functions. This privilege is defined by the `version_tokens` plugin; see Version Tokens.

- **XA_RECOVER_ADMIN**

  Enables execution of the `XA RECOVER` statement; see XA Transaction SQL Statements.

Prior to MySQL 8.0, any user could execute the `XA RECOVER` statement to discover the XID values for outstanding prepared XA transactions, possibly leading to commit or rollback of an XA transaction by a user other than the one who started it. In MySQL 8.0, `XA RECOVER` is permitted only to users who have the `XA_RECOVER_ADMIN` privilege, which is expected to be granted only to administrative users who have need for it. This might be the case, for example, for administrators of an XA application if it has crashed and it is necessary to find outstanding transactions started by the application so they can be rolled back. This privilege requirement prevents users from discovering the XID values for outstanding prepared XA transactions other than their own. It does not affect normal commit or rollback of an XA transaction because the user who started it knows its XID.

Privilege-Granting Guidelines

It is a good idea to grant to an account only those privileges that it needs. You should exercise particular caution in granting the `FILE` and administrative privileges:

- **FILE** can be abused to read into a database table any files that the MySQL server can read on the server host. This includes all world-readable files and files in the server's data directory. The table can then be accessed using `SELECT` to transfer its contents to the client host.

- **GRANT OPTION** enables users to give their privileges to other users. Two users that have different privileges and with the `GRANT OPTION` privilege are able to combine privileges.

- **ALTER** may be used to subvert the privilege system by renaming tables.

- **SHUTDOWN** can be abused to deny service to other users entirely by terminating the server.
Static Versus Dynamic Privileges

MySQL supports static and dynamic privileges:

- Static privileges are built in to the server. They are always available to be granted to user accounts and cannot be unregistered.

- Dynamic privileges can be registered and unregistered at runtime. This affects their availability: A dynamic privilege that has not been registered cannot be granted.

For example, the `SELECT` and `INSERT` privileges are static and always available, whereas a dynamic privilege becomes available only if the server component that implements it has been enabled.

The remainder of this section describes how dynamic privileges work in MySQL. The discussion uses the term “components” but applies equally to plugins.

### Note

Server administrators should be aware of which server components define dynamic privileges. For MySQL distributions, documentation of components that define dynamic privileges describes those privileges.

Third-party components may also define dynamic privileges; an administrator should understand those privileges and not install components that might conflict or compromise server operation. For example, one component conflicts with another if both define a privilege with the same name. Component developers can reduce the likelihood of this occurrence by choosing privilege names having a prefix based on the component name.

The server maintains the set of registered dynamic privileges internally in memory. Unregistration occurs at server shutdown.

Normally, a server component that defines dynamic privileges registers them when it is installed, during its initialization sequence. When uninstalled, a server component does not unregister its registered dynamic privileges. (This is current practice, not a requirement. That is, components could, but do not, unregister at any time privileges they register.)

No warning or error occurs for attempts to register an already registered dynamic privilege. Consider the following sequence of statements:

```
INSTALL COMPONENT 'my_component';
UNINSTALL COMPONENT 'my_component';
INSTALL COMPONENT 'my_component';
```
The first `INSTALL COMPONENT` statement registers any privileges defined by server component `my_component`, but `UNINSTALL COMPONENT` does not unregister them. For the second `INSTALL COMPONENT` statement, the component privileges it registers are found to be already registered, but no warnings or errors occur.

Dynamic privileges apply only at the global level. The server stores information about current assignments of dynamic privileges to user accounts in the `mysql.global_grants` system table:

- The server automatically registers privileges named in `global_grants` during server startup (unless the `--skip-grant-tables` option is given).
- The `GRANT` and `REVOKE` statements modify the contents of `global_grants`.
- Dynamic privilege assignments listed in `global_grants` are persistent. They are not removed at server shutdown.

Example: The following statement grants to user `u1` the privileges required to control replication (including Group Replication) on a slave server, and to modify system variables:

```
GRANT REPLICATION_SLAVE_ADMIN, GROUP_REPLICATION_ADMIN, BINLOG_ADMIN
ON *.* TO 'u1'@'localhost';
```

Granted dynamic privileges appear in the output from the `SHOW GRANTS` statement and the `INFORMATION_SCHEMA.USER_PRIVILEGES` table.

For `GRANT` and `REVOKE` at the global level, any named privileges not recognized as static are checked against the current set of registered dynamic privileges and granted if found. Otherwise, an error occurs to indicate an unknown privilege identifier.

For `GRANT` and `REVOKE` the meaning of `ALL [PRIVILEGES]` at the global level includes all static global privileges, as well as all currently registered dynamic privileges:

- `GRANT ALL` at the global level grants all static global privileges and all currently registered dynamic privileges. A dynamic privilege registered subsequent to execution of the `GRANT` statement is not granted retroactively to any account.
- `REVOKE ALL` at the global level revokes all granted static global privileges and all granted dynamic privileges.

The `FLUSH PRIVILEGES` statement reads the `global_grants` table for dynamic privilege assignments and registers any unregistered privileges found there.

For descriptions of the dynamic privileges provided by MySQL Server and server components included in MySQL distributions, see Section 4.2, "Privileges Provided by MySQL".

### Migrating Accounts from SUPER to Dynamic Privileges

In MySQL 8.0, many operations that previously required the `SUPER` privilege are also associated with a dynamic privilege of more limited scope. (For descriptions of these privileges, see Section 4.2, "Privileges Provided by MySQL"). Each such operation can be permitted to an account by granting the associated dynamic privilege rather than `SUPER`. This change improves security by enabling DBAs to avoid granting `SUPER` and tailor user privileges more closely to the operations permitted. `SUPER` is now deprecated and will be removed in a future version of MySQL.

When removal of `SUPER` occurs, operations that formerly required `SUPER` will fail unless accounts granted `SUPER` are migrated to the appropriate dynamic privileges. Use the following instructions to accomplish that goal so that accounts are ready prior to `SUPER` removal:

1. Execute this query to identify accounts that are granted `SUPER`:

```
SELECT GRANTEE FROM INFORMATION_SCHEMA.USER_PRIVILEGES
WHERE PRIVILEGE_TYPE = 'SUPER';
```
2. For each account identified by the preceding query, determine the operations for which it needs `SUPER`. Then grant the dynamic privileges corresponding to those operations, and revoke `SUPER`.

For example, if `'u1'@'localhost'` requires `SUPER` for binary log purging and system variable modification, these statements make the required changes to the account:

```
GRANT BINLOG_ADMIN, SYSTEM_VARIABLES_ADMIN ON *.* TO 'u1'@'localhost';
REVOKE SUPER ON *.* FROM 'u1'@'localhost';
```

After you have modified all applicable accounts, the `INFORMATION_SCHEMA` query in the first step should produce an empty result set.

### 4.3 Grant Tables

The `mysql` system database includes several grant tables that contain information about user accounts and the privileges held by them. This section describes those tables. For information about other tables in the system database, see the [mysql System Schema](#).

The discussion here describes the underlying structure of the grant tables and how the server uses their contents when interacting with clients. However, normally you do not modify the grant tables directly. Modifications occur indirectly when you use account-management statements such as `CREATE USER`, `GRANT`, and `REVOKE` to set up accounts and control the privileges available to each one. See [Account Management Statements](#). When you use such statements to perform account manipulations, the server modifies the grant tables on your behalf.

#### Note

Direct modification of grant tables using statements such as `INSERT`, `UPDATE`, or `DELETE` is discouraged and done at your own risk. The server is free to ignore rows that become malformed as a result of such modifications.

For any operation that modifies a grant table, the server checks whether the table has the expected structure and produces an error if not. To update the tables to the expected structure, perform the MySQL upgrade procedure. See [Upgrading MySQL](#).

- Grant Table Overview
- The user and db Grant Tables
- The `tables_priv` and `columns_priv` Grant Tables
- The `procs_priv` Grant Table
- The `proxies_priv` Grant Table
- The `global_grants` Grant Table
- The `default_roles` Grant Table
- The `role_edges` Grant Table
- The `password_history` Grant Table
- Grant Table Scope Column Properties
- Grant Table Privilege Column Properties

#### Grant Table Overview

These `mysql` database tables contain grant information:

- **user**: User accounts, static global privileges, and other nonprivilege columns.
Grant Table Overview

- global_grants: Dynamic global privileges.
- db: Database-level privileges.
- tables_priv: Table-level privileges.
- columns_priv: Column-level privileges.
- procs_priv: Stored procedure and function privileges.
- proxies_priv: Proxy-user privileges.
- default_roles: Default user roles.
- role_edges: Edges for role subgraphs.
- password_history: Password change history.

For information about the differences between static and dynamic global privileges, see Static Versus Dynamic Privileges.

In MySQL 8.0, grant tables use the InnoDB storage engine and are transactional. Before MySQL 8.0, grant tables used the MyISAM storage engine and were nontransactional. This change of grant table storage engine enables an accompanying change to the behavior of account-management statements such as CREATE USER or GRANT. Previously, an account-management statement that named multiple users could succeed for some users and fail for others. Now, each statement is transactional and either succeeds for all named users or rolls back and has no effect if any error occurs.

Each grant table contains scope columns and privilege columns:

- Scope columns determine the scope of each row in the tables; that is, the context in which the row applies. For example, a user table row with Host and User values of 'h1.example.net' and 'bob' applies to authenticating connections made to the server from the host h1.example.net by a client that specifies a user name of bob. Similarly, a db table row with Host, User, and Db column values of 'h1.example.net', 'bob' and 'reports' applies when bob connects from the host h1.example.net to access the reports database. The tables_priv and columns_priv tables contain scope columns indicating tables or table/column combinations to which each row applies. The procs_priv scope columns indicate the stored routine to which each row applies.

- Privilege columns indicate which privileges a table row grants; that is, which operations it permits to be performed. The server combines the information in the various grant tables to form a complete description of a user's privileges. Section 4.7, "Access Control, Stage 2: Request Verification", describes the rules for this.

In addition, a grant table may contain columns used for purposes other than scope or privilege assessment.

The server uses the grant tables in the following manner:

- The user table scope columns determine whether to reject or permit incoming connections. For permitted connections, any privileges granted in the user table indicate the user's static global privileges. Any privileges granted in this table apply to all databases on the server.

  **Caution**
  
  Because any static global privilege is considered a privilege for all databases, any static global privilege enables a user to see all database names with SHOW DATABASES or by examining the SCHEMATA table of INFORMATION_SCHEMA, except databases that have been restricted at the database level by partial revokes.

- The global_grants table lists current assignments of dynamic global privileges to user accounts. For each row, the scope columns determine which user has the privilege named in the privilege column.
The user and db Grant Tables

- The db table scope columns determine which users can access which databases from which hosts. The privilege columns determine the permitted operations. A privilege granted at the database level applies to the database and to all objects in the database, such as tables and stored programs.

- The tables_priv and columns_priv tables are similar to the db table, but are more fine-grained. They apply at the table and column levels rather than at the database level. A privilege granted at the table level applies to the table and to all its columns. A privilege granted at the column level applies only to a specific column.

- The procs_priv table applies to stored routines (stored procedures and functions). A privilege granted at the routine level applies only to a single procedure or function.

- The proxies_priv table indicates which users can act as proxies for other users and whether a user can grant the PROXY privilege to other users.

- The default_roles and role_edges tables contain information about role relationships.

- The password_history table retains previously chosen passwords to enable restrictions on password reuse. See Section 4.15, "Password Management".

The server reads the contents of the grant tables into memory when it starts. You can tell it to reload the tables by issuing a FLUSH PRIVILEGES statement or executing a mysqladmin flush-privileges or mysqladmin reload command. Changes to the grant tables take effect as indicated in Section 4.13, "When Privilege Changes Take Effect".

When you modify an account, it is a good idea to verify that your changes have the intended effect. To check the privileges for a given account, use the SHOW GRANTS statement. For example, to determine the privileges that are granted to an account with user name and host name values of bob and pc84.example.com, use this statement:

```sql
SHOW GRANTS FOR 'bob'@'pc84.example.com';
```

To display nonprivilege properties of an account, use SHOW CREATE USER:

```sql
SHOW CREATE USER 'bob'@'pc84.example.com';
```

The user and db Grant Tables

The server uses the user and db tables in the mysql database at both the first and second stages of access control (see Chapter 4, Access Control and Account Management). The columns in the user and db tables are shown here.

Table 4.3 user and db Table Columns

<table>
<thead>
<tr>
<th>Table Name</th>
<th>user</th>
<th>db</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope columns</td>
<td>Host</td>
<td>Host</td>
</tr>
<tr>
<td></td>
<td>User</td>
<td>Db</td>
</tr>
<tr>
<td>Privilege columns</td>
<td>Select_priv</td>
<td>Select_priv</td>
</tr>
<tr>
<td></td>
<td>Insert_priv</td>
<td>Insert_priv</td>
</tr>
<tr>
<td></td>
<td>Update_priv</td>
<td>Update_priv</td>
</tr>
<tr>
<td></td>
<td>Delete_priv</td>
<td>Delete_priv</td>
</tr>
<tr>
<td></td>
<td>Index_priv</td>
<td>Index_priv</td>
</tr>
<tr>
<td></td>
<td>Alter_priv</td>
<td>Alter_priv</td>
</tr>
<tr>
<td></td>
<td>Create_priv</td>
<td>Create_priv</td>
</tr>
<tr>
<td></td>
<td>Drop_priv</td>
<td>Drop_priv</td>
</tr>
<tr>
<td></td>
<td>Grant_priv</td>
<td>Grant_priv</td>
</tr>
<tr>
<td></td>
<td>Create_view_priv</td>
<td>Create_view_priv</td>
</tr>
</tbody>
</table>
### The user and db Grant Tables

<table>
<thead>
<tr>
<th>Table Name</th>
<th>user</th>
<th>db</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show_view_priv</td>
<td>Show_view_priv</td>
<td></td>
</tr>
<tr>
<td>Create_routine_priv</td>
<td>Create_routine_priv</td>
<td></td>
</tr>
<tr>
<td>Alter_routine_priv</td>
<td>Alter_routine_priv</td>
<td></td>
</tr>
<tr>
<td>Execute_priv</td>
<td>Execute_priv</td>
<td></td>
</tr>
<tr>
<td>Trigger_priv</td>
<td>Trigger_priv</td>
<td></td>
</tr>
<tr>
<td>Event_priv</td>
<td>Event_priv</td>
<td></td>
</tr>
<tr>
<td>Create_tmp_table_priv</td>
<td>Create_tmp_table_priv</td>
<td></td>
</tr>
<tr>
<td>Lock_tables_priv</td>
<td>Lock_tables_priv</td>
<td></td>
</tr>
<tr>
<td>References_priv</td>
<td>References_priv</td>
<td></td>
</tr>
<tr>
<td>Reload_priv</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shutdown_priv</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process_priv</td>
<td></td>
<td></td>
</tr>
<tr>
<td>File_priv</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Show_db_priv</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Super_priv</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Repl_slave_priv</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Repl_client_priv</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create_user_priv</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create_tablespace_priv</td>
<td>Create_tablespace_priv</td>
<td></td>
</tr>
<tr>
<td>Create_role_priv</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Drop_role_priv</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Security columns
- ssl_type
- ssl_cipher
- x509_issuer
- x509_subject
- plugin
- authentication_string
- password_expired
- password_last_changed
- password_lifetime
- account_locked
- Password_reuse_history
- Password_reuse_time
- Password_require_current
- User_attributes

#### Resource control columns
- max_questions
- max_updates
- max_connections
- max_user_connections

The **user** table `plugin` and `authentication_string` columns store authentication plugin and credential information.
The server uses the plugin named in the `plugin` column of an account row to authenticate connection attempts for the account.

The `plugin` column must be nonempty. At startup, and at runtime when `FLUSH PRIVILEGES` is executed, the server checks `user` table rows. For any row with an empty `plugin` column, the server writes a warning to the error log of this form:

```
[Warning] User entry 'user_name'@'host_name' has an empty plugin value. The user will be ignored and no one can login with this user anymore.
```

To assign a plugin to an account that is missing one, use the `ALTER USER` statement.

The `password_expired` column permits DBAs to expire account passwords and require users to reset their password. The default `password_expired` value is 'N', but can be set to 'Y' with the `ALTER USER` statement. After an account's password has been expired, all operations performed by the account in subsequent connections to the server result in an error until the user issues an `ALTER USER` statement to establish a new account password.

```
Note
Although it is possible to “reset” an expired password by setting it to its current value, it is preferable, as a matter of good policy, to choose a different password. DBAs can enforce non-reuse by establishing an appropriate password-reuse policy. See Password Reuse Policy.
```

`password_last_changed` is a `TIMESTAMP` column indicating when the password was last changed. The value is non-`NULL` only for accounts that use a MySQL built-in authentication plugin (`mysql_native_password`, `sha256_password`, or `caching_sha2_password`). The value is `NULL` for other accounts, such as those authenticated using an external authentication system.

`password_last_changed` is updated by the `CREATE USER`, `ALTER USER`, and `SET PASSWORD` statements, and by `GRANT` statements that create an account or change an account password.

`password_lifetime` indicates the account password lifetime, in days. If the password is past its lifetime (assessed using the `password_last_changed` column), the server considers the password expired when clients connect using the account. A value of `N` greater than zero means that the password must be changed every `N` days. A value of 0 disables automatic password expiration. If the value is `NULL` (the default), the global expiration policy applies, as defined by the `default_password_lifetime` system variable.

`account_locked` indicates whether the account is locked (see Section 4.19, “Account Locking”).

`Password_reuse_history` is the value of the `PASSWORD HISTORY` option for the account, or `NULL` for the default history.

`Password_reuse_time` is the value of the `PASSWORD REUSE INTERVAL` option for the account, or `NULL` for the default interval.

`Password_require_current` (added in MySQL 8.0.13) corresponds to the value of the `PASSWORD REQUIRE` option for the account, as shown by the following table.

<table>
<thead>
<tr>
<th>Password_require_current Value</th>
<th>Corresponding PASSWORD REQUIRE Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Y'</td>
<td>PASSWORD REQUIRE CURRENT</td>
</tr>
<tr>
<td>'N'</td>
<td>PASSWORD REQUIRE CURRENT OPTIONAL</td>
</tr>
<tr>
<td>NULL</td>
<td>PASSWORD REQUIRE CURRENT DEFAULT</td>
</tr>
</tbody>
</table>

`User_attributes` (added in MySQL 8.0.14) is a JSON-format column that stores account attributes not stored in other columns:
The tables_priv and columns_priv Grant Tables

- **additional_password**: The secondary password, if any. See Dual Password Support.

- **Restrictions**: Restriction lists, if any. Restrictions are added by partial-revoke operations. The attribute value is an array of elements that each have `Database` and `Restrictions` keys indicating the name of a restricted database and the applicable restrictions on it (see Section 4.12, “Privilege Restriction Using Partial Revokes”).

- **Password_locking**: The conditions for failed-login tracking and temporary account locking, if any (see Failed-Login Tracking and Temporary Account Locking). The `Password_locking` attribute is updated according to the `FAILED_LOGIN_ATTEMPTS` and `PASSWORD_LOCK_TIME` options of the CREATE USER and ALTER USER statements. The attribute value is a hash with `failed_login_attempts` and `password_lock_time_days` keys indicating the value of such options as have been specified for the account. If a key is missing, its value is implicitly 0. If a key value is implicitly or explicitly 0, the corresponding capability is disabled. This attribute was added in MySQL 8.0.19.

If no attributes apply, `User_attributes` is NULL.

Example: An account that has a secondary password and partially revoked database privileges has `additional_password` and `Restrictions` attributes in the column value:

```sql
mysql> SELECT User, Host, JSON_KEYS(User_attributes) FROM mysql.user WHERE User_attributes IS NOT NULL;
```

To determine which attributes are present, use the `JSON_KEYS()` function:

```sql
SELECT User, Host, JSON_KEYS(User_attributes) FROM mysql.user WHERE User_attributes IS NOT NULL;
```

To extract a particular attribute, such as `Restrictions`, do this:

```sql
SELECT User, Host, User_attributes->>'$.Restrictions' FROM mysql.user WHERE User_attributes->>'$.Restrictions' <> '';```

### The tables_priv and columns_priv Grant Tables

During the second stage of access control, the server performs request verification to ensure that each client has sufficient privileges for each request that it issues. In addition to the `user` and `db` grant tables, the server may also consult the `tables_priv` and `columns_priv` tables for requests that involve tables. The latter tables provide finer privilege control at the table and column levels. They have the columns shown in the following table.

#### Table 4.5 tables_priv and columns_priv Table Columns

<table>
<thead>
<tr>
<th>Table Name</th>
<th>tables_priv</th>
<th>columns_priv</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scope columns</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Host</td>
<td>Host</td>
<td></td>
</tr>
<tr>
<td>Db</td>
<td>Db</td>
<td></td>
</tr>
<tr>
<td>User</td>
<td>User</td>
<td></td>
</tr>
<tr>
<td>Table_name</td>
<td>Table_name</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Column_name</td>
<td></td>
</tr>
<tr>
<td><strong>Privilege columns</strong></td>
<td>Table_priv</td>
<td>Column_priv</td>
</tr>
<tr>
<td></td>
<td>Column_priv</td>
<td></td>
</tr>
<tr>
<td><strong>Other columns</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timestamp</td>
<td>Timestamp</td>
<td></td>
</tr>
<tr>
<td>Grantor</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The procs_priv Grant Table

The Timestamp and Grantor columns are set to the current timestamp and the CURRENT_USER value, respectively, but are otherwise unused.

The procs_priv Grant Table

For verification of requests that involve stored routines, the server may consult the procs_priv table, which has the columns shown in the following table.

Table 4.6 procs_priv Table Columns

<table>
<thead>
<tr>
<th>Table Name</th>
<th>procs_priv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope columns</td>
<td>Host</td>
</tr>
<tr>
<td></td>
<td>Db</td>
</tr>
<tr>
<td></td>
<td>User</td>
</tr>
<tr>
<td></td>
<td>Routine_name</td>
</tr>
<tr>
<td></td>
<td>Routine_type</td>
</tr>
<tr>
<td>Privilege columns</td>
<td>Proc_priv</td>
</tr>
<tr>
<td>Other columns</td>
<td>Timestamp</td>
</tr>
<tr>
<td></td>
<td>Grantor</td>
</tr>
</tbody>
</table>

The Routine_type column is an ENUM column with values of 'FUNCTION' or 'PROCEDURE' to indicate the type of routine the row refers to. This column enables privileges to be granted separately for a function and a procedure with the same name.

The Timestamp and Grantor columns are unused.

The proxies_priv Grant Table

The proxies_priv table records information about proxy accounts. It has these columns:

- Host, User: The proxy account; that is, the account that has the PROXY privilege for the proxied account.
- Proxied_host, Proxied_user: The proxied account.
- Grantor, Timestamp: Unused.
- With_grant: Whether the proxy account can grant the PROXY privilege to other accounts.

For an account to be able to grant the PROXY privilege to other accounts, it must have a row in the proxies_priv table with With_grant set to 1 and Proxied_host and Proxied_user set to indicate the account or accounts for which the privilege can be granted. For example, the 'root'@'localhost' account created during MySQL installation has a row in the proxies_priv table that enables granting the PROXY privilege for ''@'', that is, for all users and all hosts. This enables root to set up proxy users, as well as to delegate to other accounts the authority to set up proxy users. See Section 4.18, "Proxy Users".

The global_grants Grant Table

The global_grants table lists current assignments of dynamic global privileges to user accounts. The table has these columns:

- USER, HOST: The user name and host name of the account to which the privilege is granted.
- PRIV: The privilege name.
- WITH_GRANT_OPTION: Whether the account can grant the privilege to other accounts.
The default_roles Grant Table

The `default_roles` table lists default user roles. It has these columns:

- **HOST, USER**: The account or role to which the default role applies.
- **DEFAULT_ROLE_HOST, DEFAULT_ROLE_USER**: The default role.

The role_edges Grant Table

The `role_edges` table lists edges for role subgraphs. It has these columns:

- **FROM_HOST, FROM_USER**: The account that is granted a role.
- **TO_HOST, TO_USER**: The role that is granted to the account.
- **WITH_ADMIN_OPTION**: Whether the account can grant the role to and revoke it from other accounts by using `WITH ADMIN OPTION`.

The password_history Grant Table

The `password_history` table contains information about password changes. It has these columns:

- **Host, User**: The account for which the password change occurred.
- **Password_timestamp**: The time when the password change occurred.
- **Password**: The new password hash value.

The `password_history` table accumulates a sufficient number of nonempty passwords per account to enable MySQL to perform checks against both the account password history length and reuse interval. Automatic pruning of entries that are outside both limits occurs when password-change attempts occur.

Note

The empty password does not count in the password history and is subject to reuse at any time.

If an account is renamed, its entries are renamed to match. If an account is dropped or its authentication plugin is changed, its entries are removed.

Grant Table Scope Column Properties

Scope columns in the grant tables contain strings. The default value for each is the empty string. The following table shows the number of characters permitted in each column.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Maximum Permitted Characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host, Proxied_host</td>
<td>255 (60 prior to MySQL 8.0.17)</td>
</tr>
<tr>
<td>User, Proxied_user</td>
<td>32</td>
</tr>
<tr>
<td>Db</td>
<td>64</td>
</tr>
<tr>
<td>Table_name</td>
<td>64</td>
</tr>
<tr>
<td>Column_name</td>
<td>64</td>
</tr>
<tr>
<td>Routine_name</td>
<td>64</td>
</tr>
</tbody>
</table>
Host and Proxied_host values are converted to lowercase before being stored in the grant tables.

For access-checking purposes, comparisons of User, Proxied_user, authentication_string, Db, and Table_name values are case-sensitive. Comparisons of Host, Proxied_host, Column_name, and Routine_name values are not case-sensitive.

Grant Table Privilege Column Properties

The user and db tables list each privilege in a separate column that is declared as ENUM('N','Y') DEFAULT 'N'. In other words, each privilege can be disabled or enabled, with the default being disabled.

The tables_priv, columns_priv, and procs_priv tables declare the privilege columns as SET columns. Values in these columns can contain any combination of the privileges controlled by the table. Only those privileges listed in the column value are enabled.

Table 4.8 Set-Type Privilege Column Values

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Column Name</th>
<th>Possible Set Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>tables_priv</td>
<td>Table_priv</td>
<td>'Select', 'Insert', 'Update', 'Delete',</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Create', 'Drop', 'Grant', 'References',</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Index', 'Alter', 'Create View', 'Show</td>
</tr>
<tr>
<td></td>
<td></td>
<td>view', 'Trigger'</td>
</tr>
<tr>
<td>tables_priv</td>
<td>Column_priv</td>
<td>'Select', 'Insert', 'Update', 'References'</td>
</tr>
<tr>
<td>columns_priv</td>
<td>Column_priv</td>
<td>'Select', 'Insert', 'Update', 'References'</td>
</tr>
<tr>
<td>procs_priv</td>
<td>Proc_priv</td>
<td>'Execute', 'Alter Routine', 'Grant'</td>
</tr>
</tbody>
</table>

Only the user and global_grants tables specify administrative privileges, such as RELOAD, SHUTDOWN, and SYSTEM_VARIABLES_ADMIN. Administrative operations are operations on the server itself and are not database-specific, so there is no reason to list these privileges in the other grant tables. Consequently, the server need consult only the user and global_grants tables to determine whether a user can perform an administrative operation.

The FILE privilege also is specified only in the user table. It is not an administrative privilege as such, but a user's ability to read or write files on the server host is independent of the database being accessed.

4.4 Specifying Account Names

MySQL account names consist of a user name and a host name, which enables creation of distinct accounts for users with the same user name who can connect from different hosts. This section describes how to write account names, including special values and wildcard rules.

MySQL role names are similar to account names, with some differences described at Section 4.5, "Specifying Role Names".

In SQL statements such as CREATE USER, GRANT, and SET PASSWORD, account names follow these rules:

- Account name syntax is 'user_name'@'host_name'.

- An account name consisting only of a user name is equivalent to 'user_name'@'%. For example, 'me' is equivalent to 'me'@'%."
Specifying Account Names

- The user name and host name need not be quoted if they are legal as unquoted identifiers. Quotes are necessary to specify a `user_name` string containing special characters (such as space or `-`), or a `host_name` string containing special characters or wildcard characters (such as `. ` or `%`). For example, in the account name `test-user'@'%.com'`, both the user name and host name parts require quotes.

- Quote user names and host names as identifiers or as strings, using either backticks (``), single quotation marks (`'`), or double quotation marks (`"`). For string-quoting and identifier-quoting guidelines, see String Literals, and Schema Object Names.

- The user name and host name parts, if quoted, must be quoted separately. That is, write `me'@'localhost'`, not `me@localhost`. The latter is actually equivalent to `me@localhost'@'%`.

- A reference to the CURRENT_USER or CURRENT_USER() function is equivalent to specifying the current client's user name and host name literally.

MySQL stores account names in grant tables in the `mysql` system database using separate columns for the user name and host name parts:

- The `user` table contains one row for each account. The User and Host columns store the user name and host name. This table also indicates which global privileges the account has.

- Other grant tables indicate privileges an account has for databases and objects within databases. These tables have User and Host columns to store the account name. Each row in these tables associates with the account in the user table that has the same User and Host values.

- For access-checking purposes, comparisons of User values are case-sensitive. Comparisons of Host values are not case sensitive.

For additional detail about the properties of user names and host names as stored in the grant tables, such as maximum length, see Grant Table Scope Column Properties.

User names and host names have certain special values or wildcard conventions, as described following.

The user name part of an account name is either a nonblank value that literally matches the user name for incoming connection attempts, or a blank value (empty string) that matches any user name. An account with a blank user name is an anonymous user. To specify an anonymous user in SQL statements, use a quoted empty user name part, such as `''@'localhost'`.

The host name part of an account name can take many forms, and wildcards are permitted:

- A host value can be a host name or an IP address (IPv4 or IPv6). The name `localhost` indicates the local host. The IP address `127.0.0.1` indicates the IPv4 loopback interface. The IP address `::1` indicates the IPv6 loopback interface.

- The `%` and `_` wildcard characters are permitted in host name or IP address values. These have the same meaning as for pattern-matching operations performed with the LIKE operator. For example, a host value of `%` matches any host name, whereas a value of `.mysql.com` matches any host in the `mysql.com` domain. `198.51.100.%` matches any host in the 198.51.100 class C network.

Because IP wildcard values are permitted in host values (for example, `198.51.100.%` to match every host on a subnet), someone could try to exploit this capability by naming a host `198.51.100.somewhere.com`. To foil such attempts, MySQL does not perform matching on host names that start with digits and a dot. For example, if a host is named `1.2.example.com`, its name never matches the host part of account names. An IP wildcard value can match only IP addresses, not host names.

- For a host value specified as an IPv4 address, a netmask can be given to indicate how many address bits to use for the network number. Netmask notation cannot be used for IPv6 addresses.
Specifying Role Names

The syntax is *host_ip/netmask*. For example:

```
CREATE USER 'david'@'198.51.100.0/255.255.255.0';
```

This enables *david* to connect from any client host having an IP address *client_ip* for which the following condition is true:

```
client_ip & netmask = host_ip
```

That is, for the `CREATE USER` statement just shown:

```
client_ip & 255.255.255.0 = 198.51.100.0
```

IP addresses that satisfy this condition range from *198.51.100.0* to *198.51.100.255*.

A netmask typically begins with bits set to 1, followed by bits set to 0. Examples:

- *198.0.0.0/255.0.0.0*: Any host on the 198 class A network
- *198.51.100.0/255.255.0.0*: Any host on the 198.51 class B network
- *198.51.100.0/255.255.255.0*: Any host on the 198.51.100 class C network
- *198.51.100.1*: Only the host with this specific IP address

The server performs matching of host values in account names against the client host using the value returned by the system DNS resolver for the client host name or IP address. Except in the case that the account host value is specified using netmask notation, the server performs this comparison as a string match, even for an account host value given as an IP address. This means that you should specify account host values in the same format used by DNS. Here are examples of problems to watch out for:

- Suppose that a host on the local network has a fully qualified name of *host1.example.com*. If DNS returns name lookups for this host as *host1.example.com*, use that name in account host values.
- If DNS returns just *host1*, use *host1* instead.
- If DNS returns the IP address for a given host as *198.51.100.2*, that will match an account host value of *198.51.100.2* but not *198.051.100.2*. Similarly, it will match an account host pattern like *198.51.100.%* but not *198.051.100.%*.

To avoid problems like these, it is advisable to check the format in which your DNS returns host names and addresses. Use values in the same format in MySQL account names.

### 4.5 Specifying Role Names

MySQL role names refer to roles, which are named collections of privileges. For role usage examples, see Section 4.10, “Using Roles”.

Role names have syntax and semantics similar to account names; see Section 4.4, “Specifying Account Names”. As stored in the grant tables, they have the same properties as account names, which are described in Grant Table Scope Column Properties.

Role names differ from account names in these respects:

- The user part of role names cannot be blank. Thus, there is no “anonymous role” analogous to the concept of “anonymous user.”
- As for an account name, omitting the host part of a role name results in a host part of *’%’*. But unlike *’%’* in an account name, a host part of *’%’* in a role name has no wildcard properties. For example, for a name *’me’@’%’* used as a role name, the host part (’%’) is just a literal value; it has no “any host” matching property.
- Netmask notation in the host part of a role name has no significance.
An account name is permitted to be `CURRENT_USER()` in several contexts. A role name is not.

It is possible for a row in the `mysql.user` system table to serve as both an account and a role. In this case, any special user or host name matching properties do not apply in contexts for which the name is used as a role name. For example, you cannot execute the following statement with the expectation that it will set the current session roles using all roles that have a user part of `myrole` and any host name:

```sql
SET ROLE 'myrole'@'%';
```

Instead, the statement sets the active role for the session to the role with exactly the name `myrole'@'%'.

For this reason, role names are often specified using only the user name part and letting the host name part implicitly be `@'`. Specifying a role with a non-'@' host part can be useful if you intend to create a name that works both as a role an as a user account that is permitted to connect from the given host.

### 4.6 Access Control, Stage 1: Connection Verification

When you attempt to connect to a MySQL server, the server accepts or rejects the connection based on these conditions:

- Your identity and whether you can verify your identity by supplying the correct password
- Whether your account is locked or unlocked

The server checks credentials first, then account locking state. A failure for either step causes the server to deny access to you completely. Otherwise, the server accepts the connection, and then enters Stage 2 and waits for requests.

Credential checking is performed using the three `user` table scope columns (`Host`, `User`, and `authentication_string`). Locking state is recorded in the `user` table `account_locked` column. The server accepts the connection only if the `Host` and `User` columns in some `user` table row match the client host name and user name, the client supplies the password specified in that row, and the `account_locked` value is `'N'`. The rules for permissible `Host` and `User` values are given in Section 4.4, “Specifying Account Names”. Account locking can be changed with the `ALTER USER` statement.

Your identity is based on two pieces of information:

- The client host from which you connect
- Your MySQL user name

If the `User` column value is nonblank, the user name in an incoming connection must match exactly. If the `User` value is blank, it matches any user name. If the `user` table row that matches an incoming connection has a blank user name, the user is considered to be an anonymous user with no name, not a user with the name that the client actually specified. This means that a blank user name is used for all further access checking for the duration of the connection (that is, during Stage 2).

The `authentication_string` column can be blank. This is not a wildcard and does not mean that any password matches. It means that the user must connect without specifying a password. If the server authenticates a client using a plugin, the authentication method that the plugin implements may or may not use the password in the `authentication_string` column. In this case, it is possible that an external password is also used to authenticate to the MySQL server.

Nonblank `authentication_string` values in the `user` table represent encrypted passwords. MySQL does not store passwords as cleartext for anyone to see. Rather, the password supplied by a user who is attempting to connect is encrypted (using the password hashing method implemented
by the account authentication plugin). The encrypted password then is used during the connection process when checking whether the password is correct. This is done without the encrypted password ever traveling over the connection. See Section 4.1, “Account User Names and Passwords”.

From MySQL’s point of view, the encrypted password is the real password, so you should never give anyone access to it. In particular, do not give nonadministrative users read access to tables in the mysql system database.

The following table shows how various combinations of User and Host values in the user table apply to incoming connections.

<table>
<thead>
<tr>
<th>User Value</th>
<th>Host Value</th>
<th>Permissible Connections</th>
</tr>
</thead>
<tbody>
<tr>
<td>'fred'</td>
<td>'h1.example.net'</td>
<td>fred, connecting from h1.example.net</td>
</tr>
<tr>
<td>''</td>
<td>'h1.example.net'</td>
<td>Any user, connecting from h1.example.net</td>
</tr>
<tr>
<td>'fred'</td>
<td>'%'</td>
<td>fred, connecting from any host</td>
</tr>
<tr>
<td>''</td>
<td>'%'</td>
<td>Any user, connecting from any host</td>
</tr>
<tr>
<td>'fred'</td>
<td>'%.example.net'</td>
<td>fred, connecting from any host in the example.net domain</td>
</tr>
<tr>
<td>'fred'</td>
<td>'x.example.%'</td>
<td>fred, connecting from x.example.net, x.example.com, x.example.edu, and so on; this is probably not useful</td>
</tr>
<tr>
<td>'fred'</td>
<td>'198.51.100.177'</td>
<td>fred, connecting from the host with IP address 198.51.100.177</td>
</tr>
<tr>
<td>'fred'</td>
<td>'198.51.100.%'</td>
<td>fred, connecting from any host in the 198.51.100 class C subnet</td>
</tr>
<tr>
<td>'fred'</td>
<td>'198.51.100.0/255.255.255.0'</td>
<td>Same as previous example</td>
</tr>
</tbody>
</table>

It is possible for the client host name and user name of an incoming connection to match more than one row in the user table. The preceding set of examples demonstrates this: Several of the entries shown match a connection from h1.example.net by fred.

When multiple matches are possible, the server must determine which of them to use. It resolves this issue as follows:

- Whenever the server reads the user table into memory, it sorts the rows.
- When a client attempts to connect, the server looks through the rows in sorted order.
- The server uses the first row that matches the client host name and user name.

The server uses sorting rules that order rows with the most-specific Host values first. Literal host names and IP addresses are the most specific. (The specificity of a literal IP address is not affected by whether it has a netmask, so 198.51.100.13 and 198.51.100.0/255.255.255.0 are considered equally specific.) The pattern '%' means “any host” and is least specific. The empty string '' also means “any host” but sorts after '%'. Rows with the same Host value are ordered with the most-specific User values first (a blank User value means “any user” and is least specific). For rows with equally-specific Host and User values, the order is nondeterministic.

To see how this works, suppose that the user table looks like this:

<table>
<thead>
<tr>
<th>Host</th>
<th>User</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>localhost</td>
<td>root</td>
<td>...</td>
</tr>
<tr>
<td>%</td>
<td>root</td>
<td>...</td>
</tr>
<tr>
<td>%</td>
<td>jeffrey</td>
<td>...</td>
</tr>
<tr>
<td>localhost</td>
<td>root</td>
<td>...</td>
</tr>
</tbody>
</table>
When the server reads the table into memory, it sorts the rows using the rules just described. The result after sorting looks like this:

<table>
<thead>
<tr>
<th>Host</th>
<th>User</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>localhost</td>
<td>root</td>
<td>...</td>
</tr>
<tr>
<td>localhost</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%</td>
<td>jeffrey</td>
<td>...</td>
</tr>
<tr>
<td>%</td>
<td>root</td>
<td>...</td>
</tr>
</tbody>
</table>

When a client attempts to connect, the server looks through the sorted rows and uses the first match found. For a connection from `localhost` by `jeffrey`, two of the rows from the table match: the one with Host and User values of 'localhost' and '', and the one with values of '%' and 'jeffrey'. The 'localhost' row appears first in sorted order, so that is the one the server uses.

Here is another example. Suppose that the user table looks like this:

<table>
<thead>
<tr>
<th>Host</th>
<th>User</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td>jeffrey</td>
<td>...</td>
</tr>
<tr>
<td>h1.example.net</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The sorted table looks like this:

<table>
<thead>
<tr>
<th>Host</th>
<th>User</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>h1.example.net</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%</td>
<td>jeffrey</td>
<td>...</td>
</tr>
</tbody>
</table>

A connection by `jeffrey` from `h1.example.net` is matched by the first row, whereas a connection by `jeffrey` from any host is matched by the second.

**Note**

It is a common misconception to think that, for a given user name, all rows that explicitly name that user are used first when the server attempts to find a match for the connection. This is not true. The preceding example illustrates this, where a connection from `h1.example.net` by `jeffrey` is first matched not by the row containing 'jeffrey' as the User column value, but by the row with no user name. As a result, `jeffrey` is authenticated as an anonymous user, even though he specified a user name when connecting.

If you are able to connect to the server, but your privileges are not what you expect, you probably are being authenticated as some other account. To find out what account the server used to authenticate you, use the `CURRENT_USER()` function. (See Information Functions.) It returns a value in `user_name@host_name` format that indicates the User and Host values from the matching user table row. Suppose that `jeffrey` connects and issues the following query:

```sql
mysql> SELECT CURRENT_USER();
+----------------+
| CURRENT_USER() |
+----------------+
| @localhost     |
```

The result shown here indicates that the matching user table row had a blank User column value. In other words, the server is treating `jeffrey` as an anonymous user.

Another way to diagnose authentication problems is to print out the user table and sort it by hand to see where the first match is being made.
4.7 Access Control, Stage 2: Request Verification

After you establish a connection, the server enters Stage 2 of access control. For each request that you issue through that connection, the server determines what operation you want to perform, then checks whether you have sufficient privileges to do so. This is where the privilege columns in the grant tables come into play. These privileges can come from any of the `user`, `global_grants`, `db`, `tables_priv`, `columns_priv`, or `procs_priv` tables. (You may find it helpful to refer to Section 4.3, “Grant Tables”, which lists the columns present in each grant table.)

The `user` and `global_grants` tables grant global privileges. The rows in these tables for a given account indicate the account privileges that apply on a global basis no matter what the default database is. For example, if the `user` table grants you the `DELETE` privilege, you can delete rows from any table in any database on the server host. It is wise to grant privileges in the `user` table only to people who need them, such as database administrators. For other users, leave all privileges in the `user` table set to ‘N’ and grant privileges at more specific levels only (for particular databases, tables, columns, or routines). It is also possible to grant database privileges globally but use partial revokes to restrict them from being exercised on specific databases (see Section 4.12, “Privilege Restriction Using Partial Revokes”).

The `db` table grants database-specific privileges. Values in the scope columns of this table can take the following forms:

- A blank `User` value matches the anonymous user. A nonblank value matches literally; there are no wildcards in user names.
- The wildcard characters `%` and `_` can be used in the `Host` and `Db` columns. These have the same meaning as for pattern-matching operations performed with the `LIKE` operator. If you want to use either character literally when granting privileges, you must escape it with a backslash. For example, to include the underscore character ( `_` ) as part of a database name, specify it as `\_` in the `GRANT` statement.
- A '%' or blank `Host` value means “any host.”
- A '%' or blank `Db` value means “any database.”

The server reads the `db` table into memory and sorts it at the same time that it reads the `user` table. The server sorts the `db` table based on the `Host`, `Db`, and `User` scope columns. As with the `user` table, sorting puts the most-specific values first and least-specific values last, and when the server looks for matching rows, it uses the first match that it finds.

The `tables_priv`, `columns_priv`, and `procs_priv` tables grant table-specific, column-specific, and routine-specific privileges. Values in the scope columns of these tables can take the following forms:

- The wildcard characters `%` and `_` can be used in the `Host` column. These have the same meaning as for pattern-matching operations performed with the `LIKE` operator.
- A '%' or blank `Host` value means “any host.”
- The `Db`, `Table_name`, `Column_name`, and `Routine_name` columns cannot contain wildcards or be blank.

The server sorts the `tables_priv`, `columns_priv`, and `procs_priv` tables based on the `Host`, `Db`, and `User` columns. This is similar to `db` table sorting, but simpler because only the `Host` column can contain wildcards.

The server uses the sorted tables to verify each request that it receives. For requests that require administrative privileges such as `SHUTDOWN` or `RELOAD`, the server checks only the `user` and `global_privilege` tables because those are the only tables that specify administrative privileges.
Adding Accounts, Assigning Privileges, and Dropping Accounts

The server grants access if a row for the account in those tables permits the requested operation and denies access otherwise. For example, if you want to execute `mysqladmin shutdown` but your `user` table row does not grant the `SHUTDOWN` privilege to you, the server denies access without even checking the `db` table. (The latter table contains no `Shutdown_priv` column, so there is no need to check it.)

For database-related requests (`INSERT`, `UPDATE`, and so on), the server first checks the user's global privileges in the `user` table row (less any privilege restrictions imposed by partial revokes). If the row permits the requested operation, access is granted. If the global privileges in the `user` table are insufficient, the server determines the user's database-specific privileges from the `db` table:

The server looks in the `db` table for a match on the `Host`, `Db`, and `User` columns. The `Host` and `User` columns are matched to the connecting user's host name and MySQL user name. The `Db` column is matched to the database that the user wants to access. If there is no row for the `Host` and `User`, access is denied.

After determining the database-specific privileges granted by the `db` table rows, the server adds them to the global privileges granted by the `user` table. If the result permits the requested operation, access is granted. Otherwise, the server successively checks the user's table and column privileges in the `tables_priv` and `columns_priv` tables, adds those to the user's privileges, and permits or denies access based on the result. For stored-routine operations, the server uses the `procs_priv` table rather than `tables_priv` and `columns_priv`.

Expressed in boolean terms, the preceding description of how a user's privileges are calculated may be summarized like this:

```
global privileges
OR (database privileges AND host privileges)
OR table privileges
OR column privileges
OR routine privileges
```

It may not be apparent why, if the global privileges are initially found to be insufficient for the requested operation, the server adds those privileges to the database, table, and column privileges later. The reason is that a request might require more than one type of privilege. For example, if you execute an `INSERT INTO ... SELECT` statement, you need both the `INSERT` and the `SELECT` privileges. Your privileges might be such that the `user` table row grants one privilege global and the `db` table row grants the other specifically for the relevant database. In this case, you have the necessary privileges to perform the request, but the server cannot tell that from either your global or database privileges alone. It must make an access-control decision based on the combined privileges.

### 4.8 Adding Accounts, Assigning Privileges, and Dropping Accounts

To manage MySQL accounts, use the SQL statements intended for that purpose:

- `CREATE USER` and `DROP USER` create and remove accounts.
- `GRANT` and `REVOKE` assign privileges to and revoke privileges from accounts.
- `SHOW GRANTS` displays account privilege assignments.

Account-management statements cause the server to make appropriate modifications to the underlying grant tables, which are discussed in Section 4.3, “Grant Tables”.

**Note**

Direct modification of grant tables using statements such as `INSERT`, `UPDATE`, or `DELETE` is discouraged and done at your own risk. The server is free to ignore rows that become malformed as a result of such modifications.
Creating Accounts and Granting Privileges

For any operation that modifies a grant table, the server checks whether the table has the expected structure and produces an error if not. To update the tables to the expected structure, perform the MySQL upgrade procedure. See Upgrading MySQL.

Another option for creating accounts is to use the GUI tool MySQL Workbench. Also, several third-party programs offer capabilities for MySQL account administration. phpMyAdmin is one such program.

This section discusses the following topics:

- Creating Accounts and Granting Privileges
- Checking Account Privileges and Properties
- Revoking Account Privileges
- Dropping Accounts

For additional information about the statements discussed here, see Account Management Statements.

Creating Accounts and Granting Privileges

The following examples show how to use the `mysql` client program to set up new accounts. These examples assume that the MySQL root account has the `CREATE USER` privilege and all privileges that it grants to other accounts.

At the command line, connect to the server as the MySQL root user, supplying the appropriate password at the password prompt:

```
shell> mysql -u root -p
Enter password: (enter root password here)
```

After connecting to the server, you can add new accounts. The following example uses `CREATE USER` and `GRANT` statements to set up four accounts (where you see `password`, substitute an appropriate password):

```sql
CREATE USER 'finley'@'localhost'
IDENTIFIED BY 'password';
GRANT ALL
ON *.*
TO 'finley'@'localhost'
WITH GRANT OPTION;
CREATE USER 'finley'@'%.example.com'
IDENTIFIED BY 'password';
GRANT ALL
ON *.*
TO 'finley'@'%.example.com'
WITH GRANT OPTION;
CREATE USER 'admin'@'localhost'
IDENTIFIED BY 'password';
GRANT RELOAD,PROCESS
ON *.*
TO 'admin'@'localhost';
CREATE USER 'dummy'@'localhost';
```

The accounts created by those statements have the following properties:

- Two accounts have a user name of `finley`. Both are superuser accounts with full global privileges to do anything. The `'finley'@'localhost'` account can be used only when connecting from the local host. The `'finley'@'%.example.com'` account uses the `'%` wildcard in the host part, so it can be used to connect from any host in the example.com domain.

The `'finley'@'localhost'` account is necessary if there is an anonymous-user account for `localhost`. Without the `'finley'@'localhost'` account, that anonymous-user account takes
Checking Account Privileges and Properties

preference when finley connects from the local host and finley is treated as an anonymous user. The reason for this is that the anonymous-user account has a more specific Host column value than the 'finley'@$' account and thus comes earlier in the user table sort order. (For information about user table sorting, see Section 4.6, "Access Control, Stage 1: Connection Verification").

• The 'admin'@'localhost' account can be used only by admin to connect from the local host. It is granted the global RELOAD and PROCESS administrative privileges. These privileges enable the admin user to execute the mysqladmin reload, mysqladmin refresh, and mysqladmin flush-xxx commands, as well as mysqladmin processlist. No privileges are granted for accessing any databases. You could add such privileges using GRANT statements.

• The 'dummy'@'localhost' account has no password (which is insecure and not recommended). This account can be used only to connect from the local host. No privileges are granted. It is assumed that you will grant specific privileges to the account using GRANT statements.

The previous example grants privileges at the global level. The next example creates three accounts and grants them access at lower levels; that is, to specific databases or objects within databases. Each account has a user name of custom, but the host name parts differ:

```
CREATE USER 'custom'@'localhost'
  IDENTIFIED BY 'password';
GRANT ALL
  ON bankaccount.*
  TO 'custom'@'localhost';
CREATE USER 'custom'@'host47.example.com'
  IDENTIFIED BY 'password';
GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP
  ON expenses.*
  TO 'custom'@'host47.example.com';
CREATE USER 'custom'@'%.example.com'
  IDENTIFIED BY 'password';
GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP
  ON customer.addresses
  TO 'custom'@'%.example.com';
```

The three accounts can be used as follows:

• The 'custom'@'localhost' account has all database-level privileges to access the bankaccount database. The account can be used to connect to the server only from the local host.

• The 'custom'@'host47.example.com' account has specific database-level privileges to access the expenses database. The account can be used to connect to the server only from the host host47.example.com.

• The 'custom'@'%.example.com' account has specific table-level privileges to access the addresses table in the customer database, from any host in the example.com domain. The account can be used to connect to the server from all machines in the domain due to use of the % wildcard character in the host part of the account name.

Checking Account Privileges and Properties

To see the privileges for an account, use SHOW GRANTS:

```
mysql> SHOW GRANTS FOR 'admin'@'localhost';
+-----------------------------------------------------+
| Grants for admin@localhost                          |
| GRANT RELOAD, PROCESS ON *.* TO 'admin'@'localhost' |
+-----------------------------------------------------+
```

To see nonprivilege properties for an account, use SHOW CREATE USER:

```
mysql> SET print_identified_with_as_hex = ON;
mysql> SHOW CREATE USER 'admin'@'localhost'\G
65
```
### Revoking Account Privileges

To revoke account privileges, use the `REVOKE` statement. Privileges can be revoked at different levels, just as they can be granted at different levels.

Revoke global privileges:

```sql
REVOKE ALL ON *.* FROM 'finley'@'%.example.com';
REVOKE RELOAD ON *.* FROM 'admin'@'localhost';
```

Revoke database-level privileges:

```sql
REVOKE CREATE,DROP ON expenses.* FROM 'custom'@'host47.example.com';
```

Revoke table-level privileges:

```sql
REVOKE INSERT,UPDATE,DELETE ON customer.addresses FROM 'custom'@'%.example.com';
```

To check the effect of privilege revocation, use `SHOW GRANTS`:

```sql
mysql> SHOW GRANTS FOR 'admin'@'localhost';
+---------------------------------------------+
| Grants for admin@localhost                 |
| +---------------------------------------------+
| GRANT PROCESS ON *.* TO 'admin'@'localhost' |
+---------------------------------------------+
```

### Dropping Accounts

To remove an account, use the `DROP USER` statement. For example, to drop some of the accounts created previously:

```sql
DROP USER 'finley'@'localhost';
DROP USER 'finley'@'%.example.com';
DROP USER 'admin'@'localhost';
DROP USER 'dummy'@'localhost';
```

### 4.9 Reserved Accounts

One part of the MySQL installation process is data directory initialization (see Section 3.1, “Initializing the Data Directory”). During data directory initialization, MySQL creates user accounts that should be considered reserved:

- `'root'@'localhost'`: Used for administrative purposes. This account has all privileges, is a system account, and can perform any operation.
Strictly speaking, this account name is not reserved, in the sense that some installations rename the root account to something else to avoid exposing a highly privileged account with a well-known name.

- 'mysql.sys'@'localhost': Used as the DEFINER for sys schema objects. Use of the mysql.sys account avoids problems that occur if a DBA renames or removes the root account. This account is locked so that it cannot be used for client connections.

- 'mysql.session'@'localhost': Used internally by plugins to access the server. This account is locked so that it cannot be used for client connections. The account is a system account.

- 'mysql.infoschema'@'localhost': Used as the DEFINER for INFORMATION_SCHEMA views. Use of the mysql.infoschema account avoids problems that occur if a DBA renames or removes the root account. This account is locked so that it cannot be used for client connections.

### 4.10 Using Roles

A MySQL role is a named collection of privileges. Like user accounts, roles can have privileges granted to and revoked from them.

A user account can be granted roles, which grants to the account the privileges associated with each role. This enables assignment of sets of privileges to accounts and provides a convenient alternative to granting individual privileges, both for conceptualizing desired privilege assignments and implementing them.

The following list summarizes role-management capabilities provided by MySQL:

- **CREATE ROLE** and **DROP ROLE** create and remove roles.
- **GRANT** and **REVOKE** assign privileges to revoke privileges from user accounts and roles.
- **SHOW GRANTS** displays privilege and role assignments for user accounts and roles.
- **SET DEFAULT ROLE** specifies which account roles are active by default.
- **SET ROLE** changes the active roles within the current session.
- The **CURRENT_ROLE()** function displays the active roles within the current session.
- The mandatory_roles and activate_all_roles_on_login system variables enable defining mandatory roles and automatic activation of granted roles when users log in to the server.

For descriptions of individual role-manipulation statements (including the privileges required to use them), see Account Management Statements. The following discussion provides examples of role usage. Unless otherwise specified, SQL statements shown here should be executed using a MySQL account with sufficient administrative privileges, such as the root account.

- Creating Roles and Granting Privileges to Them
- Defining Mandatory Roles
- Checking Role Privileges
- Activating Roles
- Revoking Roles or Role Privileges
- Dropping Roles
- User and Role Interchangeability
Creating Roles and Granting Privileges to Them

Consider this scenario:

- An application uses a database named `app_db`.
- Associated with the application, there can be accounts for developers who create and maintain the application, and for users who interact with it.
- Developers need full access to the database. Some users need only read access, others need read/write access.

To avoid granting privileges individually to possibly many user accounts, create roles as names for the required privilege sets. This makes it easy to grant the required privileges to user accounts, by granting the appropriate roles.

To create the roles, use the `CREATE ROLE` statement:

```
CREATE ROLE 'app_developer', 'app_read', 'app_write';
```

Role names are much like user account names and consist of a user part and host part in `user_name'@'host_name'` format. The host part, if omitted, defaults to `%'`. The user and host parts can be unquoted unless they contain special characters such as `-` or `%`. Unlike account names, the user part of role names cannot be blank. For additional information, see Section 4.5, “Specifying Role Names”.

To assign privileges to the roles, execute `GRANT` statements using the same syntax as for assigning privileges to user accounts:

```
GRANT ALL ON app_db.* TO 'app_developer';
GRANT SELECT ON app_db.* TO 'app_read';
GRANT INSERT, UPDATE, DELETE ON app_db.* TO 'app_write';
```

Now suppose that initially you require one developer account, two user accounts that need read-only access, and one user account that needs read/write access. Use `CREATE USER` to create the accounts:

```
CREATE USER 'dev1'@'localhost' IDENTIFIED BY 'dev1pass';
CREATE USER 'read_user1'@'localhost' IDENTIFIED BY 'read_user1pass';
CREATE USER 'read_user2'@'localhost' IDENTIFIED BY 'read_user2pass';
CREATE USER 'rw_user1'@'localhost' IDENTIFIED BY 'rw_user1pass';
```

To assign each user account its required privileges, you could use `GRANT` statements of the same form as just shown, but that requires enumerating individual privileges for each user. Instead, use an alternative `GRANT` syntax that permits granting roles rather than privileges:

```
GRANT 'app_developer' TO 'dev1'@'localhost';
GRANT 'app_read' TO 'read_user1'@'localhost', 'read_user2'@'localhost';
GRANT 'app_read', 'app_write' TO 'rw_user1'@'localhost';
```

The `GRANT` statement for the `rw_user1` account grants the read and write roles, which combine to provide the required read and write privileges.

The `GRANT` syntax for granting roles to an account differs from the syntax for granting privileges: There is an `ON` clause to assign privileges, whereas there is no `ON` clause to assign roles. Because the syntaxes are distinct, you cannot mix assigning privileges and roles in the same statement. (It is permitted to assign both privileges and roles to an account, but you must use separate `GRANT` statements, each with syntax appropriate to what is to be granted.) As of MySQL 8.0.16, roles cannot be granted to anonymous users.

A role when created is locked, has no password, and is assigned the default authentication plugin. (These role attributes can be changed later with the `ALTER USER` statement, by users who have the `GLOBAL CREATE USER` privilege.)
Defining Mandatory Roles

While locked, a role cannot be used to authenticate to the server. If unlocked, a role can be used to authenticate. This is because roles and users are both authorization identifiers with much in common and little to distinguish them. See also User and Role Interchangeability.

**Defining Mandatory Roles**

It is possible to specify roles as mandatory by naming them in the value of the `mandatory_roles` system variable. The server treats a mandatory role as granted to all users, so that it need not be granted explicitly to any account.

To specify mandatory roles at server startup, define `mandatory_roles` in your server `my.cnf` file:

```sql
[mysqld]
mandatory_roles='role1,role2@localhost,r3@%.example.com'
```

To set and persist `mandatory_roles` at runtime, use a statement like this:

```sql
SET PERSIST mandatory_roles = 'role1,role2@localhost,r3@%.example.com';
```

`SET PERSIST` sets the value for the running MySQL instance. It also saves the value, causing it to be used for subsequent server restarts. To change the value for the running MySQL instance without saving it for subsequent restarts, use the `GLOBAL` keyword rather than `PERSIST`. See SET Syntax for Variable Assignment.

Setting `mandatory_roles` requires the `ROLE_ADMIN` privilege, in addition to the `SYSTEM_VARIABLES_ADMIN` or `SUPER` privilege normally required to set a global system variable.

Mandatory roles, like explicitly granted roles, do not take effect until activated (see Activating Roles). At login time, role activation occurs for all granted roles if the `activate_all_roles_on_login` system variable is enabled, or for roles that are set as default roles otherwise. At runtime, `SET ROLE` activates roles.

Roles named in the value of `mandatory_roles` cannot be revoked with `REVOKE` or dropped with `DROP ROLE` or `DROP USER`.

To prevent sessions from being made system sessions by default, a role that has the `SYSTEM_USER` privilege cannot be listed in the value of the `mandatory_roles` system variable:

- If `mandatory_roles` is assigned a role at startup that has the `SYSTEM_USER` privilege, the server writes a message to the error log and exits.
- If `mandatory_roles` is assigned a role at runtime that has the `SYSTEM_USER` privilege, an error occurs and the `mandatory_roles` value remains unchanged.

If a role named in `mandatory_roles` is not present in the `mysql.user` system table, the role is not granted to users. When the server attempts role activation for a user, it does not treat the nonexistent role as mandatory and writes a warning to the error log. If the role is created later and thus becomes valid, `FLUSH PRIVILEGES` may be necessary to cause the server to treat it as mandatory.

`SHOW GRANTS` displays mandatory roles according to the rules described in SHOW GRANTS Statement.

**Checking Role Privileges**

To verify the privileges assigned to an account, use `SHOW GRANTS`. For example:

```sql
mysql> SHOW GRANTS FOR 'dev1'@'localhost';
+-------------------------------------------------+
<table>
<thead>
<tr>
<th>Grants for dev1@localhost</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRANT USAGE ON <em>.</em> TO <code>dev1</code>@<code>localhost</code></td>
</tr>
<tr>
<td>GRANT <code>app_developer</code>@<code>%</code> TO <code>dev1</code>@<code>localhost</code></td>
</tr>
</tbody>
</table>
+-------------------------------------------------+
```
Activating Roles

However, that shows each granted role without “expanding” it to the privileges the role represents. To show role privileges as well, add a `USING` clause naming the granted roles for which to display privileges:

```
mysql> SHOW GRANTS FOR 'dev1'@'localhost' USING 'app_developer';
+-----------------------------------------------+
| Grants for dev1@localhost                    |
+-----------------------------------------------+
| GRANT USAGE ON *.* TO `dev1`@`localhost`     |
| GRANT ALL PRIVILEGES ON `app_db`.* TO `dev1`@`localhost` |
| GRANT `app_developer`@`%` TO `dev1`@`localhost` |
+-----------------------------------------------+
```

Verify each other type of user similarly:

```
mysql> SHOW GRANTS FOR 'read_user1'@'localhost' USING 'app_read';
+--------------------------------+|
| Grants for read_user1@localhost |
+--------------------------------+|
| GRANT USAGE ON *.* TO `read_user1`@`localhost` |
| GRANT SELECT ON `app_db`.* TO `read_user1`@`localhost` |
| GRANT `app_read`@`%` TO `read_user1`@`localhost` |
+--------------------------------+|
```

```
mysql> SHOW GRANTS FOR 'rw_user1'@'localhost' USING 'app_read', 'app_write';
+------------------------------------------------------------------+
| Grants for rw_user1@localhost                                    |
+------------------------------------------------------------------+
| GRANT USAGE ON *.* TO `rw_user1`@`localhost`                     |
| GRANT SELECT, INSERT, UPDATE, DELETE ON `app_db`.* TO `rw_user1`@`localhost` |
| GRANT `app_read`@`%`,`app_write`@`%` TO `rw_user1`@`localhost`    |
+------------------------------------------------------------------+
```

SHOW GRANTS displays mandatory roles according to the rules described in SHOW GRANTS Statement.

Activating Roles

Roles granted to a user account can be active or inactive within account sessions. If a granted role is active within a session, its privileges apply; otherwise, they do not. To determine which roles are active within the current session, use the `CURRENT_ROLE()` function.

By default, granting a role to an account or naming it in the `mandatory_roles` system variable value does not automatically cause the role to become active within account sessions. For example, because thus far in the preceding discussion no `rw_user1` roles have been activated, if you connect to the server as `rw_user1` and invoke the `CURRENT_ROLE()` function, the result is `NONE` (no active roles):

```
mysql> SELECT CURRENT_ROLE();
+----------------+
| CURRENT_ROLE() |
+----------------+
| NONE           |
+----------------+
```

To specify which roles should become active each time a user connects to the server and authenticates, use `SET DEFAULT ROLE`. To set the default to all assigned roles for each account created earlier, use this statement:

```
SET DEFAULT ROLE ALL TO
`dev1`@`localhost`,
`read_user1`@`localhost`,
`read_user2`@`localhost`,
`rw_user1`@`localhost`;
```

Now if you connect as `rw_user1`, the initial value of `CURRENT_ROLE()` reflects the new default role assignments:

```
mysql> SELECT CURRENT_ROLE();
+----------------+
| CURRENT_ROLE() |
+----------------+
```
To cause all explicitly granted and mandatory roles to be automatically activated when users connect to the server, enable the `activate_all_roles_on_login` system variable. By default, automatic role activation is disabled.

Within a session, a user can execute `SET ROLE` to change the set of active roles. For example, for `rw_user1`:

```sql
mysql> SET ROLE NONE; SELECT CURRENT_ROLE();
+----------------+
| CURRENT_ROLE() |
+----------------+
| NONE           |
+----------------+
mysql>
mysql> SET ROLE ALL EXCEPT 'app_write'; SELECT CURRENT_ROLE();
+----------------+
| CURRENT_ROLE() |
+----------------+
| `app_read`@`%` |
+----------------+
mysql>
mysql> SET ROLE DEFAULT; SELECT CURRENT_ROLE();
+--------------------------------+
| CURRENT_ROLE()                 |
+--------------------------------+
| `app_read`@`%`,`app_write`@`%` |
+--------------------------------+
```

The first `SET ROLE` statement deactivates all roles. The second makes `rw_user1` effectively read only. The third restores the default roles.

The effective user for stored program and view objects is subject to the `DEFINER` and `SQL SECURITY` attributes, which determine whether execution occurs in invoker or definer context (see Stored Object Access Control):

- Stored program and view objects that execute in invoker context execute with the roles that are active within the current session.
- Stored program and view objects that execute in definer context execute with the default roles of the user named in their `DEFINER` attribute. If `activate_all_roles_on_login` is enabled, such objects execute with all roles granted to the `DEFINER` user, including mandatory roles. For stored programs, if execution should occur with roles different from the default, the program body should execute `SET ROLE` to activate the required roles.

## Revoking Roles or Role Privileges

Just as roles can be granted to an account, they can be revoked from an account:

```sql
REVOKE role FROM user;
```

Roles named in the `mandatory_roles` system variable value cannot be revoked.

`REVOKE` can also be applied to a role to modify the privileges granted to it. This affects not only the role itself, but any account granted that role. Suppose that you want to temporarily make all application users read only. To do this, use `REVOKE` to revoke the modification privileges from the `app_write` role:

```sql
REVOKE INSERT, UPDATE, DELETE ON app_db.* FROM 'app_write';
```

As it happens, that leaves the role with no privileges at all, as can be seen using `SHOW GRANTS` (which demonstrates that this statement can be used with roles, not just users):
Dropping Roles

Because revoking privileges from a role affects the privileges for any user who is assigned the modified role, `rw_user1` now has no table modification privileges (INSERT, UPDATE, and DELETE are no longer present):

```
mysql> SHOW GRANTS FOR 'rw_user1'@'localhost'
USING 'app_read', 'app_write';
+----------------------------------------------------------------+
| Grants for rw_user1@localhost                                  |
| GRANT USAGE ON *.* TO `rw_user1`@`localhost`                   |
| GRANT SELECT ON `app_db`.* TO `rw_user1`@`localhost`           |
| GRANT `app_read`@`%`,`app_write`@`%` TO `rw_user1`@`localhost` |
+----------------------------------------------------------------+
```

In effect, the `rw_user1` read/write user has become a read-only user. This also occurs for any other accounts that are granted the `app_write` role, illustrating how use of roles makes it unnecessary to modify privileges for individual accounts.

To restore modification privileges to the role, simply re-grant them:

```
GRANT INSERT, UPDATE, DELETE ON app_db.* TO 'app_write';
```

Now `rw_user1` again has modification privileges, as do any other accounts granted the `app_write` role.

**Dropping Roles**

To drop roles, use `DROP ROLE`:

```
DROP ROLE 'app_read', 'app_write';
```

Dropping a role revokes it from every account to which it was granted.

Roles named in the `mandatory_roles` system variable value cannot be dropped.

**User and Role Interchangeability**

As has been hinted at earlier for `SHOW GRANTS`, which displays grants for user accounts or roles, and accounts and roles can be used interchangeably.

One difference between roles and users is that `CREATE ROLE` creates an authorization identifier that is locked by default, whereas `CREATE USER` creates an authorization identifier that is unlocked by default. However, distinction is not immutable because a user with appropriate privileges can lock or unlock roles or users after they have been created.

If a database administrator has a preference that a specific authorization identifier must be a role, a name scheme can be used to communicate this intention. For example, you could use a `r_` prefix for all authorization identifiers that you intend to be roles and nothing else.

Another difference between roles and users lies in the privileges available for administering them:

- The `CREATE ROLE` and `DROP ROLE` privileges enable only use of the `CREATE ROLE` and `DROP ROLE` statements, respectively.

- The `CREATE USER` privilege enables use of the `ALTER USER`, `CREATE ROLE`, `CREATE USER`, `DROP ROLE`, `DROP USER`, `RENAME USER`, and `REVOKE ALL PRIVILEGES` statements.
Thus, the `CREATE ROLE` and `DROP ROLE` privileges are not as powerful as `CREATE USER` and may be granted to users who should only be permitted to create and drop roles, and not perform more general account manipulation.

With regard to privileges and interchangeability of users and roles, you can treat a user account like a role and grant that account to another user or a role. The effect is to grant the account's privileges and roles to the other user or role.

This set of statements demonstrates that you can grant a user to a user, a role to a user, a user to a role, or a role to a role:

```sql
CREATE USER 'u1';
CREATE ROLE 'r1';
GRANT SELECT ON db1.* TO 'u1';
GRANT SELECT ON db2.* TO 'r1';
CREATE USER 'u2';
CREATE ROLE 'r2';
GRANT 'u1', 'r1' TO 'u2';
GRANT 'u1', 'r1' TO 'r2';
```

The result in each case is to grant to the grantee object the privileges associated with the granted object. After executing those statements, each of `u2` and `r2` have been granted privileges from a user (`u1`) and a role (`r1`):

```sql
mysql> SHOW GRANTS FOR 'u2' USING 'u1', 'r1';
+-------------------------------------+
| Grants for u2@%                     |
+-------------------------------------+
| GRANT USAGE ON *.* TO `u2`@`%`       |
| GRANT SELECT ON `db1`.* TO `u2`@`%` |
| GRANT SELECT ON `db2`.* TO `u2`@`%` |
| GRANT `u1`@`%`,`r1`@`%` TO `u2`@`%` |
+-------------------------------------+
mysql>
```

```sql
mysql> SHOW GRANTS FOR 'r2' USING 'u1', 'r1';
+-------------------------------------+
| Grants for r2@%                     |
+-------------------------------------+
| GRANT USAGE ON *.* TO `r2`@`%`       |
| GRANT SELECT ON `db1`.* TO `r2`@`%` |
| GRANT SELECT ON `db2`.* TO `r2`@`%` |
| GRANT `u1`@`%`,`r1`@`%` TO `r2`@`%` |
+-------------------------------------+
```

The preceding example is illustrative only, but interchangeability of user accounts and roles has practical application, such as in the following situation: Suppose that a legacy application development project began before the advent of roles in MySQL, so all user accounts associated with the project are granted privileges directly (rather than granted privileges by virtue of being granted roles). One of these accounts is a developer account that was originally granted privileges as follows:

```sql
CREATE USER 'old_app_dev'@'localhost' IDENTIFIED BY 'old_app_devpass';
GRANT ALL ON old_app.* TO 'old_app_dev'@'localhost';
```

If this developer leaves the project, it becomes necessary to assign the privileges to another user, or perhaps multiple users if development activities have expanded. Here are some ways to deal with the issue:

- **Without using roles:** Change the account password so the original developer cannot use it, and have a new developer use the account instead:
  ```sql
  ALTER USER 'old_app_dev'@'localhost' IDENTIFIED BY 'new_password';
  ```

- **Using roles:** Lock the account to prevent anyone from using it to connect to the server:
  ```sql
  ALTER USER 'old_app_dev'@'localhost' ACCOUNT LOCK;
  ```

Then treat the account as a role. For each developer new to the project, create a new account and grant to it the original developer account:
CREATE USER 'new_app_dev1'@'localhost' IDENTIFIED BY 'new_password';
GRANT 'old_app_dev'@'localhost' TO 'new_app_dev1'@'localhost';

The effect is to assign the original developer account privileges to the new account.

### 4.11 Account Categories

As of MySQL 8.0.16, MySQL incorporates the concept of user account categories, based on the `SYSTEM_USER` privilege.

- **System and Regular Accounts**
- **Operations Affected by the SYSTEM_USER Privilege**
- **System and Regular Sessions**
- **Protecting System Accounts Against Manipulation by Regular Accounts**

#### System and Regular Accounts

MySQL incorporates the concept of user account categories, with system and regular users distinguished according to whether they have the `SYSTEM_USER` privilege:

- A user with the `SYSTEM_USER` privilege is a system user.
- A user without the `SYSTEM_USER` privilege is a regular user.

The `SYSTEM_USER` privilege has an effect on the accounts to which a given user can apply its other privileges, as well as whether the user is protected from other accounts:

- A system user can modify both system and regular accounts. That is, a user who has the appropriate privileges to perform a given operation on regular accounts is enabled by possession of `SYSTEM_USER` to also perform the operation on system accounts. A system account can be modified only by system users with appropriate privileges, not by regular users.

- A regular user with appropriate privileges can modify regular accounts, but not system accounts. A regular account can be modified by both system and regular users with appropriate privileges.

If a user has the appropriate privileges to perform a given operation on regular accounts, `SYSTEM_USER` enables the user to also perform the operation on system accounts. `SYSTEM_USER` does not imply any other privilege, so the ability to perform a given account operation remains predicated on possession of any other required privileges. For example, if a user can grant the `SELECT` and `UPDATE` privileges to regular accounts, then with `SYSTEM_USER` the user can also grant `SELECT` and `UPDATE` to system accounts.

The distinction between system and regular accounts enables better control over certain account administration issues by protecting accounts that have the `SYSTEM_USER` privilege from accounts that do not have the privilege. For example, the `CREATE USER` privilege enables not only creation of new accounts, but modification and removal of existing accounts. Without the system user concept, a user who has the `CREATE USER` privilege can modify or drop any existing account, including the `root` account. The concept of system user enables restricting modifications to the `root` account (itself a system account) so they can be made only by system users. Regular users with the `CREATE USER` privilege can still modify or drop existing accounts, but only regular accounts.

#### Operations Affected by the SYSTEM_USER Privilege

The `SYSTEM_USER` privilege affects these operations:

- Account manipulation.
Account manipulation includes creating and dropping accounts, granting and revoking privileges, changing account authentication characteristics such as credentials or authentication plugin, and changing other account characteristics such as password expiration policy.

The **SYSTEM_USER** privilege is required to manipulate system accounts using account-management statements such as `CREATE USER` and `GRANT`. To prevent an account from modifying system accounts this way, make it a regular account by not granting it the **SYSTEM_USER** privilege. (However, to fully protect system accounts against regular accounts, you must also withhold modification privileges for the `mysql` system schema from regular accounts. See [Protecting System Accounts Against Manipulation by Regular Accounts](#).

- **Killing current sessions and statements executing within them.**

  To kill a session or statement that is executing with the **SYSTEM_USER** privilege, your own session must have the **SYSTEM_USER** privilege, in addition to any other required privilege (`CONNECTION_ADMIN` or `SUPER`).

  Prior to MySQL 8.0.16, `CONNECTION_ADMIN` or `SUPER` are sufficient to kill any session or statement.

- **Setting the **DEFINER** attribute for stored objects.**

  To set the **DEFINER** attribute for a stored object to an account that has the **SYSTEM_USER** privilege, you must have the **SYSTEM_USER** privilege, in addition to any other required privilege (`SET_USER_ID` or `SUPER`).

  Prior to MySQL 8.0.16, `SET_USER_ID` or `SUPER` are sufficient to specify any **DEFINER** value for stored objects.

- **Specifying mandatory roles.**

  A role that has the **SYSTEM_USER** privilege cannot be listed in the value of the `mandatory_roles` system variable.

  Prior to MySQL 8.0.16, any role can be listed in `mandatory_roles`.

---

**System and Regular Sessions**

Sessions executing within the server are distinguished as system or regular sessions, similar to the distinction between system and regular users:

- A session that possesses the **SYSTEM_USER** privilege is a system session.

- A session that does not possess the **SYSTEM_USER** privilege is a regular session.

A regular session is able to perform only operations permitted to regular users. A system session is additionally able to perform operations permitted only to system users.

The privileges possessed by a session are those granted directly to its underlying account, plus those granted to all roles currently active within the session. Thus, a session may be a system session because its account has been granted the **SYSTEM_USER** privilege directly, or because the session has activated a role that has the **SYSTEM_USER** privilege. Roles granted to an account that are not active within the session do not affect session privileges.

Because activating and deactivating roles can change the privileges possessed by sessions, a session may change from a regular session to a system session or vice versa. If a session activates or deactivates a role that has the **SYSTEM_USER** privilege, the appropriate change between regular and system session takes place immediately, for that session only:

- If a regular session activates a role with the **SYSTEM_USER** privilege, the session becomes a system session.
Protecting System Accounts Against Manipulation by Regular Accounts

• If a system session deactivates a role with the `SYSTEM_USER` privilege, the session becomes a regular session, unless some other role with the `SYSTEM_USER` privilege remains active.

These operations have no effect on existing sessions:

• If the `SYSTEM_USER` privilege is granted to or revoked from an account, existing sessions for the account do not change between regular and system sessions. The grant or revoke operation affects only sessions for subsequent connections by the account.

• Statements executed by a stored object invoked within a session execute with the system or regular status of the parent session, even if the object `DEFINER` attribute names a system account.

Because role activation affects only sessions and not accounts, granting a role that has the `SYSTEM_USER` privilege to a regular account does not protect that account against regular users. The role protects only sessions for the account in which the role has been activated, and protects the session only against being killed by regular sessions.

Protecting System Accounts Against Manipulation by Regular Accounts

Account manipulation includes creating and dropping accounts, granting and revoking privileges, changing account authentication characteristics such as credentials or authentication plugin, and changing other account characteristics such as password expiration policy.

Account manipulation can be done two ways:

• By using account-management statements such as `CREATE USER` and `GRANT`. This is the preferred method.

• By direct grant-table modification using statements such as `INSERT` and `UPDATE`. This method is discouraged but possible for users with the appropriate privileges on the `mysql` system schema that contains the grant tables.

To fully protect system accounts against modification by a given account, make it a regular account and do not grant it modification privileges for the `mysql` schema:

• The `SYSTEM_USER` privilege is required to manipulate system accounts using account-management statements. To prevent an account from modifying system accounts this way, make it a regular account by not granting `SYSTEM_USER` to it. This includes not granting `SYSTEM_USER` to any roles granted to the account.

• Privileges for the `mysql` schema enable manipulation of system accounts through direct modification of the grant tables, even if the modifying account is a regular account. To restrict unauthorized direct modification of system accounts by a regular account, do not grant modification privileges for the `mysql` schema to the account (or any roles granted to the account). If a regular account must have global privileges that apply to all schemas, `mysql` schema modifications can be prevented using privilege restrictions imposed using partial revokes. See Section 4.12, “Privilege Restriction Using Partial Revokes”.

Note

Unlike withholding the `SYSTEM_USER` privilege, which prevents an account from modifying system accounts but not regular accounts, withholding `mysql` schema privileges prevents an account from modifying system accounts as well as regular accounts. This should not be an issue because, as mentioned, direct grant-table modification is discouraged.

Suppose that you want to create a user `u1` who has all privileges on all schemas, except that `u1` should be a regular user without the ability to modify system accounts. Assuming that the `partial_revokes` system variable is enabled, configure `u1` as follows:

```sql
CREATE USER u1 IDENTIFIED BY 'password';
GRANT ALL ON *.* TO u1 WITH GRANT OPTION;
```
Privilege Restriction Using Partial Revokes

-- GRANT ALL includes SYSTEM_USER, so at this point
-- u1 can manipulate system or regular accounts
REVOKE SYSTEM_USER ON *.* FROM u1;
-- Revoking SYSTEM_USER makes u1 a regular user;
-- now u1 can use account-management statements
-- to manipulate only regular accounts
REVOKE ALL ON mysql.* FROM u1;
-- This partial revoke prevents u1 from directly
-- modifying grant tables to manipulate accounts

To prevent all mysql system schema access by an account, revoke all its privileges on the mysql schema, as just shown. It is also possible to permit partial mysql schema access, such as read-only access. The following example creates an account that has SELECT, INSERT, UPDATE, and DELETE privileges globally for all schemas, but only SELECT for the mysql schema:

CREATE USER u2 IDENTIFIED BY 'password';
GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO u2;
REVOKE INSERT, UPDATE, DELETE ON mysql.* FROM u2;

Another possibility is to revoke all mysql schema privileges but grant access to specific mysql tables or columns. This can be done even with a partial revoke on mysql. The following statements enable read-only access to u1 within the mysql schema, but only for the db table and the Host and User columns of the user table:

CREATE USER u3 IDENTIFIED BY 'password';
GRANT ALL ON *.* TO u3;
REVOKE ALL ON mysql.* FROM u3;
GRANT SELECT ON mysql.db TO u3;
GRANT SELECT(Host,User) ON mysql.user TO u3;

4.12 Privilege Restriction Using Partial Revokes

Prior to MySQL 8.0.16, it is not possible to grant privileges that apply globally except for certain schemas. As of MySQL 8.0.16, that is possible if the partial_revokes system variable is enabled. Specifically, for users who have privileges at the global level, partial_revokes enables privileges for specific schemas to be revoked while leaving the privileges in place for other schemas. Privilege restrictions thus imposed may be useful for administration of accounts that have global privileges but should not be permitted to access certain schemas. For example, it is possible to permit an account to modify any table except those in the mysql system schema.

- Using Partial Revokes
- Partial Revokes Versus Explicit Schema Grants
- Disabling Partial Revokes
- Partial Revokes and Replication

Note
For brevity, CREATE USER statements shown here do not include passwords.
For production use, always assign account passwords.

Using Partial Revokes

The partial_revokes system variable controls whether privilege restrictions can be placed on accounts. By default, partial_revokes is disabled and attempts to partially revoke global privileges produce an error:

code}

To permit the REVOKE operation, enable partial_revokes:
Using Partial Revokes

SET PERSIST partial_revokes = ON;

**SET PERSIST** sets the value for the running MySQL instance. It also saves the value, causing it to be used for subsequent server restarts. To change the value for the running MySQL instance without saving it for subsequent restarts, use the **GLOBAL** keyword rather than **PERSIST**. See **SET Syntax for Variable Assignment**.

With **partial_revokes** enabled, the partial revoke succeeds:

```sql
mysql> REVOKE INSERT ON world.* FROM u1;
```

```sql
mysql> SHOW GRANTS FOR u1;
```

<table>
<thead>
<tr>
<th>Grants for u1@%</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRANT SELECT, INSERT ON <em>.</em> TO <code>u1</code>@<code>%</code></td>
</tr>
<tr>
<td>REVOKE INSERT ON <code>world</code>.* FROM <code>u1</code>@<code>%</code></td>
</tr>
</tbody>
</table>

**SHOW GRANTS** lists partial revokes as **REVOKE** statements in its output. The result indicates that **u1** has global **SELECT** and **INSERT** privileges, except that **INSERT** cannot be exercised for tables in the **world** schema. That is, access by **u1** to **world** tables is read only.

The server records privilege restrictions implemented through partial revokes in the **mysql.user** system table. If an account has partial revokes, its **User_attributes** column value has a **Restrictions** attribute:

```sql
mysql> SELECT User, Host, User_attributes->>'$.Restrictions'
FROM mysql.user
WHERE User_attributes->>'$.Restrictions' <> '';
```

<table>
<thead>
<tr>
<th>User</th>
<th>Host</th>
<th>User_attributes-&gt;&gt;'$.Restrictions'</th>
</tr>
</thead>
<tbody>
<tr>
<td>u1</td>
<td>%</td>
<td></td>
</tr>
</tbody>
</table>

```
ends here
```

**Note**

Although partial revokes can be imposed for any schema, privilege restrictions on the **mysql** system schema in particular are useful as part of a strategy for preventing regular accounts from modifying system accounts. See **Protecting System Accounts Against Manipulation by Regular Accounts**.

Partial revoke operations are subject to these conditions:

- Partial revokes must name the schema literally. Schema names that contain the % or _ SQL wildcard characters (for example, **myschema%**) are not permitted.

- It is possible to use partial revokes to place restrictions on nonexistent schemas, but only if the revoked privilege is granted globally. If a privilege is not granted globally, revoking it for a nonexistent schema produces an error.

- Partial revokes apply at the schema level only. You cannot use partial revokes for privileges that apply only globally (such as **FILE** or **BINLOG_ADMIN**), or for table, column, or routine privileges.

As mentioned previously, partial revokes of schema-level privileges appear in **SHOW GRANTS** output as **REVOKE** statements. This differs from how **SHOW GRANTS** represents “plain” schema-level privileges:

- When granted, schema-level privileges are represented by their own **GRANT** statements in the output:

```sql
mysql> CREATE USER u1;
mysql> GRANT UPDATE ON mysql.* TO u1;
mysql> GRANT DELETE ON world.* TO u1;
mysql> SHOW GRANTS FOR u1;
```

<table>
<thead>
<tr>
<th>Grants for u1@%</th>
</tr>
</thead>
<tbody>
<tr>
<td>-----------------</td>
</tr>
</tbody>
</table>

ends here
When revoked, schema-level privileges simply disappear from the output. They do not appear as `REVOKE` statements:

```sql
mysql> REVOKE UPDATE ON mysql.* FROM u1;
mysql> REVOKE DELETE ON world.* FROM u1;
mysql> SHOW GRANTS FOR u1;
+------------------------------------------+
<table>
<thead>
<tr>
<th>Grants for u1@%</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRANT SELECT ON <em>.</em> TO <code>u1</code>@<code>%</code></td>
</tr>
</tbody>
</table>
+------------------------------------------+
```

When a user grants a privilege, any restriction the grantor has on the privilege is inherited by the grantee, unless the grantee already has the privilege without the restriction. Consider the following two users, one of whom has the global `SELECT` privilege:

```sql
CREATE USER u1, u2;
GRANT SELECT ON *.* TO u2;
```

Suppose that an administrative user `admin` has a global but partially revoked `SELECT` privilege:

```sql
mysql> CREATE USER admin;
mysql> GRANT SELECT ON *.* TO admin WITH GRANT OPTION;
mysql> REVOKE SELECT ON mysql.* FROM admin;
mysql> SHOW GRANTS FOR admin;
+------------------------------------------+
<table>
<thead>
<tr>
<th>Grants for admin@%</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRANT SELECT ON <em>.</em> TO <code>admin</code>@<code>%</code> WITH GRANT OPTION</td>
</tr>
<tr>
<td>REVOKE SELECT ON <code>mysql</code>.* FROM <code>admin</code>@<code>%</code></td>
</tr>
</tbody>
</table>
+------------------------------------------+
```

If `admin` grants `SELECT` globally to `u1` and `u2`, the result differs for each user:

- If `admin` grants `SELECT` globally to `u1`, who has no `SELECT` privilege to begin with, `u1` inherits the `admin` privilege restriction:

  ```sql
  mysql> GRANT SELECT ON *.* TO u1;
  mysql> SHOW GRANTS FOR u1;
  +------------------------------------------+
<table>
<thead>
<tr>
<th>Grants for u1@%</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRANT SELECT ON <em>.</em> TO <code>u1</code>@<code>%</code></td>
</tr>
<tr>
<td>REVOKE SELECT ON <code>mysql</code>.* FROM <code>u1</code>@<code>%</code></td>
</tr>
</tbody>
</table>
  +------------------------------------------+
  ```

- On the other hand, `u2` already holds a global `SELECT` privilege without restriction. `GRANT` can only add to a grantee's existing privileges, not reduce them, so if `admin` grants `SELECT` globally to `u2`, `u2` does not inherit the `admin` restriction:

  ```sql
  mysql> GRANT SELECT ON *.* TO u2;
  mysql> SHOW GRANTS FOR u2;
  +------------------------------------------+
<table>
<thead>
<tr>
<th>Grants for u2@%</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRANT SELECT ON <em>.</em> TO <code>u2</code>@<code>%</code></td>
</tr>
</tbody>
</table>
  +------------------------------------------+
  ```

If a `GRANT` statement includes an `AS user` clause, the privilege restrictions applied are those on the user/role combination specified by the clause, rather than those on the user who executes the statement. For information about the `AS` clause, see `GRANT Statement`.

Restrictions on new privileges granted to an account are added to any existing restrictions for that account:
Using Partial Revokes

```
mysql> CREATE USER u1;
mysql> GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO u1;
mysql> REVOKE INSERT ON mysql.* FROM u1;
mysql> SHOW GRANTS FOR u1;
+---------------------------------------------------------+
| Grants for u1@%                                         |
| GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO `u1`@`%`  |
| REVOKE INSERT ON `mysql`.* FROM `u1`@`%`                |
+---------------------------------------------------------+
mysql> REVOKE DELETE, UPDATE ON db2.* FROM u1;
mysql> SHOW GRANTS FOR u1;
+---------------------------------------------------------+
| Grants for u1@%                                         |
| GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO `u1`@`%`  |
| REVOKE DELETE, UPDATE ON `db2`.* FROM `u1`@`%`          |
| REVOKE INSERT ON `mysql`.* FROM `u1`@`%`                |
+---------------------------------------------------------+
```

Aggregation of privilege restrictions applies both when privileges are partially revoked explicitly (as just shown) and when restrictions are inherited implicitly from the user who executes the statement or the user mentioned in an AS user clause.

If an account has a privilege restriction on a schema:

- The account cannot grant to other accounts a privilege on the restricted schema or any object within it.
- Another account that does not have the restriction can grant privileges to the restricted account for the restricted schema or objects within it. Suppose that an unrestricted user executes these statements:

```
CREATE USER u1;
GRANT SELECT, INSERT, UPDATE ON *.* TO u1;
REVOKE SELECT, INSERT, UPDATE ON mysql.* FROM u1;
GRANT SELECT ON mysql.user TO u1;          -- grant table privilege
GRANT SELECT(Host,User) ON mysql.db TO u1; -- grant column privileges
```

The resulting account has these privileges, with the ability to perform limited operations within the restricted schema:

```
mysql> SHOW GRANTS FOR u1;
+----------------------------------------------------------+
| Grants for u1@%                                           |
| GRANT SELECT, INSERT, UPDATE ON *.* TO `u1`@`%`           |
| REVOKE SELECT, INSERT, UPDATE ON `mysql`.* FROM `u1`@`%`  |
| GRANT SELECT(`Host`, `User`) ON `mysql`.`db` TO `u1`@`%`  |
| GRANT SELECT ON `mysql`.`user` TO `u1`@`%`                |
+----------------------------------------------------------+
```

If an account has a restriction on a global privilege, the restriction is removed by any of these actions:

- Granting the privilege globally to the account by an account that has no restriction on the privilege.
- Granting the privilege at the schema level.
- Revoking the privilege globally.

Consider a user u1 who holds several privileges globally, but with restrictions on INSERT, UPDATE and DELETE:

```
mysql> CREATE USER u1;
mysql> GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO u1;
mysql> REVOKE INSERT, UPDATE, DELETE ON mysql.* FROM u1;
mysql> SHOW GRANTS FOR u1;
+---------------------------------------------------------+
| Grants for u1@%                                         |
| GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO `u1`@`%`  |
| REVOKE INSERT, UPDATE, DELETE ON `mysql`.* FROM `u1`@`%` |
+---------------------------------------------------------+
```
Using Partial Revokes

<table>
<thead>
<tr>
<th>Grants for u1@%</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRANT SELECT, INSERT, UPDATE, DELETE ON <code>*</code>.* TO <code>u1</code>@<code>%</code></td>
</tr>
<tr>
<td>REVOKE INSERT, UPDATE, DELETE ON <code>mysql</code>.* FROM <code>u1</code>@<code>%</code></td>
</tr>
</tbody>
</table>

Granting a privilege globally to u1 from an account with no restriction removes the privilege restriction. For example, to remove the INSERT restriction:

```
mysql> GRANT INSERT ON `*`.* TO u1;
mysql> SHOW GRANTS FOR u1;
```

Granting a privilege at the schema level to u1 removes the privilege restriction. For example, to remove the UPDATE restriction:

```
mysql> GRANT UPDATE ON mysql.* TO u1;
mysql> SHOW GRANTS FOR u1;
```

Revoking a global privilege removes the privilege, including any restrictions on it. For example, to remove the DELETE restriction (at the cost of removing all DELETE access):

```
mysql> REVOKE DELETE ON `*`.* FROM u1;
mysql> SHOW GRANTS FOR u1;
```

If an account has a privilege at both the global and schema levels, you must revoke it at the schema level twice to effect a partial revoke. Suppose that u1 has these privileges, where INSERT is held both globally and on the world schema:

```
mysql> CREATE USER u1;
mysql> GRANT SELECT, INSERT ON `*`.* TO u1;
mysql> GRANT INSERT ON world.* TO u1;
mysql> SHOW GRANTS FOR u1;
```

Revoking INSERT on world revokes the schema-level privilege (SHOW GRANTS no longer displays the schema-level GRANT statement):

```
mysql> REVOKE INSERT ON world.* FROM u1;
mysql> SHOW GRANTS FOR u1;
```

Revoking INSERT on world again performs a partial revoke of the global privilege (SHOW GRANTS now includes a schema-level REVOKE statement):

```
```
Partial Revokes Versus Explicit Schema Grants

To provide access to accounts for some schemas but not others, partial revokes provide an alternative to the approach of explicitly granting schema-level access without granting global privileges. The two approaches have different advantages and disadvantages.

Granting schema-level privileges and not global privileges:

- Adding a new schema: The schema is inaccessible to existing accounts by default. For any account to which the schema should be accessible, the DBA must grant schema-level access.
- Adding a new account: The DBA must grant schema-level access for each schema to which the account should have access.

Granting global privileges in conjunction with partial revokes:

- Adding a new schema: The schema is accessible to existing accounts that have global privileges. For any such account to which the schema should be inaccessible, the DBA must add a partial revoke.
- Adding a new account: The DBA must grant the global privileges, plus a partial revoke on each restricted schema.

The approach that uses explicit schema-level grant is more convenient for accounts for which access is limited to a few schemas. The approach that uses partial revokes is more convenient for accounts with broad access to all schemas except a few.

Disabling Partial Revokes

Once enabled, `partial_revokes` cannot be disabled if any account has privilege restrictions. If any such account exists, disabling `partial_revokes` fails:

- For attempts to disable `partial_revokes` at startup, the server logs an error message and enables `partial_revokes`.
- For attempts to disable `partial_revokes` at runtime, an error occurs and the `partial_revokes` value remains unchanged.

To disable `partial_revokes` when restrictions exist, the restrictions first must be removed:

1. Determine which accounts have partial revokes:

   ```sql
   SELECT User, Host, User_attributes->>'$.Restrictions'
   FROM mysql.user WHERE User_attributes->>'$.Restrictions' <> ''';
   ```

2. For each such account, remove its privilege restrictions. Suppose that the previous step shows account `u1` to have these restrictions:

   ```json
   ["Database": "world", "Privileges": ["INSERT", "DELETE"]
   ```

   Restriction removal can be done various ways:

   - Grant the privileges globally, without restrictions:
GRANT INSERT, DELETE ON *.* TO u1;

- Grant the privileges at the schema level:
  GRANT INSERT, DELETE ON world.* TO u1;

- Revoke the privileges globally (assuming that they are no longer needed):
  REVOKE INSERT, DELETE ON *.* FROM u1;

- Remove the account itself (assuming that it is no longer needed):
  DROP USER u1;

After all privilege restrictions are removed, it is possible to disable partial revokes:

SET PERSIST partial_revokes = OFF;

### Partial Revokes and Replication

In replication scenarios, if `partial_revokes` is enabled on any host, it must be enabled on all hosts. Otherwise, `REVOKE` statements to partially revoke a global privilege do not have the same effect for all hosts on which replication occurs, potentially resulting in replication inconsistencies or errors.

#### 4.13 When Privilege Changes Take Effect

If the `mysqld` server is started without the `--skip-grant-tables` option, it reads all grant table contents into memory during its startup sequence. The in-memory tables become effective for access control at that point.

If you modify the grant tables indirectly using an account-management statement, the server notices these changes and loads the grant tables into memory again immediately. Account-management statements are described in [Account Management Statements](#). Examples include `GRANT`, `REVOKE`, `SET PASSWORD`, and `RENAME USER`.

If you modify the grant tables directly using statements such as `INSERT`, `UPDATE`, or `DELETE` (which is not recommended), the changes have no effect on privilege checking until you either tell the server to reload the tables or restart it. Thus, if you change the grant tables directly but forget to reload them, the changes have *no effect* until you restart the server. This may leave you wondering why your changes seem to make no difference!

To tell the server to reload the grant tables, perform a flush-privileges operation. This can be done by issuing a `FLUSH PRIVILEGES` statement or by executing a `mysqladmin flush-privileges` or `mysqladmin reload` command.

A grant table reload affects privileges for each existing client session as follows:

- Table and column privilege changes take effect with the client's next request.
- Database privilege changes take effect the next time the client executes a `USE db_name` statement.

**Note**

Client applications may cache the database name; thus, this effect may not be visible to them without actually changing to a different database.

- Global privileges and passwords are unaffected for a connected client. These changes take effect only in sessions for subsequent connections.

Changes to the set of active roles within a session take effect immediately, for that session only. The `SET ROLE` statement performs session role activation and deactivation (see [SET ROLE Statement](#)).
Assigning Account Passwords

If the server is started with the `--skip-grant-tables` option, it does not read the grant tables or implement any access control. Any user can connect and perform any operation, which is insecure. To cause a server thus started to read the tables and enable access checking, flush the privileges.

### 4.14 Assigning Account Passwords

Required credentials for clients that connect to the MySQL server can include a password. This section describes how to assign passwords for MySQL accounts.

MySQL stores credentials in the `user` table in the `mysql` system database. Operations that assign or modify passwords are permitted only to users with the `CREATE USER` privilege, or, alternatively, privileges for the `mysql` database (INSERT privilege to create new accounts, UPDATE privilege to modify existing accounts). If the `read_only` system variable is enabled, use of account-modification statements such as `CREATE USER` or `ALTER USER` additionally requires the `CONNECTION_ADMIN` or `SUPER` privilege.

The discussion here summarizes syntax only for the most common password-assignment statements. For complete details on other possibilities, see CREATE USER Statement, ALTER USER Statement, and SET PASSWORD Statement.

MySQL uses plugins to perform client authentication; see Section 4.17, "Pluggable Authentication". In password-assigning statements, the authentication plugin associated with an account performs any hashing required of a cleartext password specified. This enables MySQL to obfuscate passwords prior to storing them in the `mysql.user` system table. For the statements described here, MySQL automatically hashes the password specified. There are also syntax for `CREATE USER` and `ALTER USER` that permits hashed values to be specified literally. For details, see the descriptions of those statements.

To assign a password when you create a new account, use `CREATE USER` and include an `IDENTIFIED BY` clause:

```
CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'password';
```

`CREATE USER` also supports syntax for specifying the account authentication plugin. See CREATE USER Statement.

To assign or change a password for an existing account, use the `ALTER USER` statement with an `IDENTIFIED BY` clause:

```
ALTER USER 'jeffrey'@'localhost' IDENTIFIED BY 'password';
```

If you are not connected as an anonymous user, you can change your own password without naming your own account literally:

```
ALTER USER USER() IDENTIFIED BY 'password';
```

To change an account password from the command line, use the `mysqladmin` command:

```
mysqladmin -u user_name -h host_name password "password"
```

The account for which this command sets the password is the one with a row in the `mysql.user` system table that matches `user_name` in the `User` column and the client host `from which you connect` in the `Host` column.

---

**Warning**

Setting a password using `mysqladmin` should be considered insecure. On some systems, your password becomes visible to system status programs such as `ps` that may be invoked by other users to display command lines. MySQL clients typically overwrite the command-line password argument with zeros during their initialization sequence. However, there is still a brief interval during which the value is visible. Also, on some systems this overwriting strategy is
ineffective and the password remains visible to `ps`. (SystemV Unix systems and perhaps others are subject to this problem.)

If you are using MySQL Replication, be aware that, currently, a password used by a replication slave as part of a `CHANGE MASTER TO` statement is effectively limited to 32 characters in length; if the password is longer, any excess characters are truncated. This is not due to any limit imposed by the MySQL Server generally, but rather is an issue specific to MySQL Replication. (For more information, see Bug #43439.)

4.15 Password Management

MySQL supports these password-management capabilities:

- Password expiration, to require passwords to be changed periodically.
- Password reuse restrictions, to prevent old passwords from being chosen again.
- Password verification, to require that password changes also specify the current password to be replaced.
- Dual passwords, to enable clients to connect using either a primary or secondary password.
- Password strength assessment, to require strong passwords.
- Random password generation, as an alternative to requiring explicit administrator-specified literal passwords.
- Password failure tracking, to enable temporary account locking after too many consecutive incorrect-password login failures.

The following sections describe these capabilities, except password strength assessment, which is implemented using the `validate_password` component and is described in Section 6.3, "The Password Validation Component".

- Internal Versus External Credentials Storage
- Password Expiration Policy
- Password Reuse Policy
- Password Verification-Required Policy
- Dual Password Support
- Random Password Generation
- Failed-Login Tracking and Temporary Account Locking

Important

MySQL implements password-management capabilities using tables in the `mysql` system database. If you upgrade MySQL from an earlier version, your system tables might not be up to date. In that case, the server writes messages similar to these to the error log during the startup process (the exact numbers may vary):

```
[ERROR] Column count of mysql.user is wrong. Expected 49, found 47. The table is probably corrupted
[Warning] ACL table mysql.password_history missing. Some operations may fail.
```

To correct the issue, perform the MySQL upgrade procedure. See Upgrading MySQL. Until this is done, password changes are not possible.
Internal Versus External Credentials Storage

Some authentication plugins store account credentials internally to MySQL, in the `mysql.user` system table:

- `mysql_native_password`
- `caching_sha2_password`
- `sha256_password`

Most discussion in this section applies to such authentication plugins because most password-management capabilities described here are based on internal credentials storage handled by MySQL itself. Other authentication plugins store account credentials externally to MySQL. For accounts that use plugins that perform authentication against an external credentials system, password management must be handled externally against that system as well.

The exception is that the options for failed-login tracking and temporary account locking apply to all accounts, not just accounts that use internal credentials storage, because MySQL is able to assess the status of login attempts for any account no matter whether it uses internal or external credentials storage.

For information about individual authentication plugins, see Section 6.1, “Authentication Plugins”.

Password Expiration Policy

MySQL enables database administrators to expire account passwords manually, and to establish a policy for automatic password expiration. Expiration policy can be established globally, and individual accounts can be set to either defer to the global policy or override the global policy with specific per-account behavior.

To expire an account password manually, use the `ALTER USER` statement:

```
ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE;
```

This operation marks the password expired in the corresponding row in the `mysql.user` system table.

Password expiration according to policy is automatic and is based on password age, which for a given account is assessed from the date and time of its most recent password change. The `mysql.user` system table indicates for each account when its password was last changed, and the server automatically treats the password as expired at client connection time if its age is greater than its permitted lifetime. This works with no explicit manual password expiration.

To establish automatic password-expiration policy globally, use the `default_password_lifetime` system variable. Its default value is 0, which disables automatic password expiration. If the value of `default_password_lifetime` is a positive integer \( N \), it indicates the permitted password lifetime, such that passwords must be changed every \( N \) days.

Examples:

- To establish a global policy that passwords have a lifetime of approximately six months, start the server with these lines in a server `my.cnf` file:

  ```
  [mysqld]
  default_password_lifetime=180
  ```

- To establish a global policy such that passwords never expire, set `default_password_lifetime` to 0:

  ```
  [mysqld]
  default_password_lifetime=0
  ```

- `default_password_lifetime` can also be set and persisted at runtime:
Password Expiration Policy

SET PERSIST default_password_lifetime = 180;
SET PERSIST default_password_lifetime = 0;

SET PERSIST sets the value for the running MySQL instance. It also saves the value to be used for subsequent server restarts; see SET Syntax for Variable Assignment. To change the value for the running MySQL instance without saving it for subsequent restarts, use the GLOBAL keyword rather than PERSIST.

The global password-expiration policy applies to all accounts that have not been set to override it. To establish policy for individual accounts, use the PASSWORD EXPIRE option of the CREATE USER and ALTER USER statements. See CREATE USER Statement, and ALTER USER Statement.

Example account-specific statements:

• Require the password to be changed every 90 days:

```sql
CREATE USER 'jeffrey'@'localhost' PASSWORD EXPIRE INTERVAL 90 DAY;
ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE INTERVAL 90 DAY;
```

This expiration option overrides the global policy for all accounts named by the statement.

• Disable password expiration:

```sql
CREATE USER 'jeffrey'@'localhost' PASSWORD EXPIRE NEVER;
ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE NEVER;
```

This expiration option overrides the global policy for all accounts named by the statement.

• Defer to the global expiration policy for all accounts named by the statement:

```sql
CREATE USER 'jeffrey'@'localhost' PASSWORD EXPIRE DEFAULT;
ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE DEFAULT;
```

When a client successfully connects, the server determines whether the account password has expired:

• The server checks whether the password has been manually expired.

• Otherwise, the server checks whether the password age is greater than its permitted lifetime according to the automatic password expiration policy. If so, the server considers the password expired.

If the password is expired (whether manually or automatically), the server either disconnects the client or restricts the operations permitted to it (see Section 4.16, “Server Handling of Expired Passwords”). Operations performed by a restricted client result in an error until the user establishes a new account password:

```sql
mysql> SELECT 1;
ERROR 1820 (HY000): You must reset your password using ALTER USER statement before executing this statement.
```

```sql
mysql> ALTER USER USER() IDENTIFIED BY 'password';
Query OK, 0 rows affected (0.01 sec)
```

```sql
mysql> SELECT 1;
+---+
| 1 |
+---+
1 row in set (0.00 sec)
```

After the client resets the password, the server restores normal access for the session, as well as for subsequent connections that use the account. It is also possible for an administrative user to reset the account password, but any existing restricted sessions for that account remain restricted. A client using the account must disconnect and reconnect before statements can be executed successfully.
Password Reuse Policy

MySQL enables restrictions to be placed on reuse of previous passwords. Reuse restrictions can be established based on number of password changes, time elapsed, or both. Reuse policy can be established globally, and individual accounts can be set to either defer to the global policy or override the global policy with specific per-account behavior.

The password history for an account consists of passwords it has been assigned in the past. MySQL can restrict new passwords from being chosen from this history:

- If an account is restricted on the basis of number of password changes, a new password cannot be chosen from a specified number of the most recent passwords. For example, if the minimum number of password changes is set to 3, a new password cannot be the same as any of the most recent 3 passwords.

- If an account is restricted based on time elapsed, a new password cannot be chosen from passwords in the history that are newer than a specified number of days. For example, if the password reuse interval is set to 60, a new password must not be among those previously chosen within the last 60 days.

  Note
  The empty password does not count in the password history and is subject to reuse at any time.

To establish password-reuse policy globally, use the `password_history` and `password_reuse_interval` system variables.

Examples:

- To prohibit reusing any of the last 6 passwords or passwords newer than 365 days, put these lines in the server `my.cnf` file:

  ```
  [mysqld]
  password_history=6
  password_reuse_interval=365
  ```

- To set and persist the variables at runtime, use statements like this:

  ```
  SET PERSIST password_history = 6;
  SET PERSIST password_reuse_interval = 365;
  ```

  `SET PERSIST` sets the value for the running MySQL instance. It also saves the value to be used for subsequent server restarts; see `SET Syntax for Variable Assignment`. To change the value for the running MySQL instance without saving it for subsequent restarts, use the `GLOBAL` keyword rather than `PERSIST`.

The global password-reuse policy applies to all accounts that have not been set to override it. To establish policy for individual accounts, use the `PASSWORD HISTORY` and `PASSWORD REUSE INTERVAL` options of the `CREATE USER` and `ALTER USER` statements. See `CREATE USER Statement`, and `ALTER USER Statement`.

Example account-specific statements:

- Require a minimum of 5 password changes before permitting reuse:
Password Verification-Required Policy

As of MySQL 8.0.13, it is possible to require that attempts to change an account password be verified by specifying the current password to be replaced. This enables DBAs to prevent users from changing a password without proving that they know the current password. Such changes could otherwise occur, for example, if one user walks away from a terminal session temporarily without logging out, and a malicious user uses the session to change the original user's MySQL password. This can have unfortunate consequences:

- The original user becomes unable to access MySQL until the account password is reset by an administrator.
- Until the password reset occurs, the malicious user can access MySQL with the benign user's changed credentials.

Password-verification policy can be established globally, and individual accounts can be set to either defer to the global policy or override the global policy with specific per-account behavior.

For each account, its `mysql.user` row indicates whether there is an account-specific setting requiring verification of the current password for password change attempts. The setting is established by the `PASSWORD REQUIRE` option of the `CREATE USER` and `ALTER USER` statements:

- If the account setting is `PASSWORD REQUIRE CURRENT`, password changes must specify the current password.
- If the account setting is `PASSWORD REQUIRE CURRENT OPTIONAL`, password changes may but need not specify the current password.
- If the account setting is `PASSWORD REQUIRE CURRENT DEFAULT`, the `password_require_current` system variable determines the verification-required policy for the account.
• If `password_require_current` is enabled, password changes must specify the current password.

• If `password_require_current` is disabled, password changes may but need not specify the current password.

In other words, if the account setting is not `PASSWORD REQUIRE CURRENT DEFAULT`, the account setting takes precedence over the global policy established by the `password_require_current` system variable. Otherwise, the account defers to the `password_require_current` setting.

By default, password verification is optional: `password_require_current` is disabled and accounts created with no `PASSWORD REQUIRE` option default to `PASSWORD REQUIRE CURRENT DEFAULT`.

The following table shows how per-account settings interact with `password_require_current` system variable values to determine account password verification-required policy.

### Table 4.9 Password-Verification Policy

<table>
<thead>
<tr>
<th>Per-Account Setting</th>
<th><code>password_require_current System Variable</code></th>
<th>Password Changes Require Current Password?</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>PASSWORD REQUIRE CURRENT</code></td>
<td>OFF</td>
<td>Yes</td>
</tr>
<tr>
<td><code>PASSWORD REQUIRE CURRENT</code></td>
<td>ON</td>
<td>Yes</td>
</tr>
<tr>
<td><code>PASSWORD REQUIRE CURRENT</code></td>
<td>OFF</td>
<td>No</td>
</tr>
<tr>
<td><code>PASSWORD REQUIRE CURRENT</code></td>
<td>ON</td>
<td>No</td>
</tr>
<tr>
<td><code>PASSWORD REQUIRE CURRENT</code></td>
<td>OFF</td>
<td>No</td>
</tr>
<tr>
<td><code>PASSWORD REQUIRE CURRENT</code></td>
<td>ON</td>
<td>Yes</td>
</tr>
</tbody>
</table>

---

**Note**

Privileged users can change any account password without specifying the current password, regardless of the verification-required policy. A privileged user is one who has the global `CREATE USER` privilege or the `UPDATE` privilege for the `mysql` system database.

To establish password-verification policy globally, use the `password_require_current` system variable. Its default value is `OFF`, so it is not required that account password changes specify the current password.

**Examples:**

• To establish a global policy that password changes must specify the current password, start the server with these lines in a server `my.cnf` file:

```
[mysqld]
password_require_current=ON
```

• To set and persist `password_require_current` at runtime, use a statement such as one of these:

```
SET PERSIST password_require_current = ON;
SET PERSIST password_require_current = OFF;
```

`SET PERSIST` sets the value for the running MySQL instance. It also saves the value to be used for subsequent server restarts; see `SET Syntax for Variable Assignment`. To change the value for the
running MySQL instance without saving it for subsequent restarts, use the `GLOBAL` keyword rather than `PERSIST`.

The global password verification-required policy applies to all accounts that have not been set to override it. To establish policy for individual accounts, use the `PASSWORD REQUIRE` options of the `CREATE USER` and `ALTER USER` statements. See `CREATE USER Statement`, and `ALTER USER Statement`.

Example account-specific statements:

- **Require that password changes specify the current password:**

```sql
CREATE USER 'jeffrey'@'localhost' PASSWORD REQUIRE CURRENT;
ALTER USER 'jeffrey'@'localhost' PASSWORD REQUIRE CURRENT;
```

This verification option overrides the global policy for all accounts named by the statement.

- **Do not require that password changes specify the current password (the current password may but need not be given):**

```sql
CREATE USER 'jeffrey'@'localhost' PASSWORD REQUIRE CURRENT OPTIONAL;
ALTER USER 'jeffrey'@'localhost' PASSWORD REQUIRE CURRENT OPTIONAL;
```

This verification option overrides the global policy for all accounts named by the statement.

- **Defer to the global password verification-required policy for all accounts named by the statement:**

```sql
CREATE USER 'jeffrey'@'localhost' PASSWORD REQUIRE CURRENT DEFAULT;
ALTER USER 'jeffrey'@'localhost' PASSWORD REQUIRE CURRENT DEFAULT;
```

Verification of the current password comes into play when a user changes a password using the `ALTER USER` or `SET PASSWORD` statement. The examples use `ALTER USER`, which is preferred over `SET PASSWORD`, but the principles described here are the same for both statements.

In password-change statements, a `REPLACE` clause specifies the current password to be replaced. Examples:

- **Change the current user’s password:**

```sql
ALTER USER USER() IDENTIFIED BY 'auth_string' REPLACE 'current_auth_string';
```

- **Change a named user’s password:**

```sql
ALTER USER 'jeffrey'@'localhost'
    IDENTIFIED BY 'auth_string'
    REPLACE 'current_auth_string';
```

- **Change a named user’s authentication plugin and password:**

```sql
ALTER USER 'jeffrey'@'localhost'
    IDENTIFIED WITH caching_sha2_password BY 'auth_string'
    REPLACE 'current_auth_string';
```

The `REPLACE` clause works like this:

- **`REPLACE` must be given if password changes for the account are required to specify the current password, as verification that the user attempting to make the change actually knows the current password.**

- **`REPLACE` is optional if password changes for the account may but need not specify the current password.**

- **If `REPLACE` is specified, it must specify the correct current password, or an error occurs. This is true even if `REPLACE` is optional.**
Dual Password Support

As of MySQL 8.0.14, user accounts are permitted to have dual passwords, designated as primary and secondary passwords. Dual-password capability makes it possible to seamlessly perform credential changes in scenarios like this:

- A system has a large number of MySQL servers, possibly involving replication.
- Multiple applications connect to different MySQL servers.
- Periodic credential changes must be made to the account or accounts used by the applications to connect to the servers.

Consider how a credential change must be performed in the preceding type of scenario when an account is permitted only a single password. In this case, there must be close cooperation in the timing of when the account password change is made and propagated throughout all servers, and when all applications that use the account are updated to use the new password. This process may involve downtime during which servers or applications are unavailable.

With dual passwords, credential changes can be made more easily, in phases, without requiring close cooperation, and without downtime:

1. For each affected account, establish a new primary password on the servers, retaining the current password as the secondary password. This enables servers to recognize either the primary or secondary password for each account, while applications can continue to connect to the servers using the same password as previously (which is now the secondary password).
2. After the password change has propagated to all servers, modify applications that use any affected account to connect using the account primary password.
3. After all applications have been migrated from the secondary passwords to the primary passwords, the secondary passwords are no longer needed and can be discarded. After this change has propagated to all servers, only the primary password for each account can be used to connect. The credential change is now complete.

MySQL implements dual-password capability with syntax that saves and discards secondary passwords:

- The `RETAIN CURRENT PASSWORD` clause for the `ALTER USER` and `SET PASSWORD` statements saves an account current password as its secondary password when you assign a new primary password.
- The `DISCARD OLD PASSWORD` clause for `ALTER USER` discards an account secondary password, leaving only the primary password.

Suppose that, for the previously described credential-change scenario, an account named 'appuser1'@'host1.example.com' is used by applications to connect to servers, and that the account password is to be changed from 'password_a' to 'password_b'.

To perform this change of credentials, use `ALTER USER` as follows:

1. On each server that is not a replication slave, establish 'password_b' as the new `appuser1` primary password, retaining the current password as the secondary password:

   ```
   ALTER USER 'appuser1'@'host1.example.com'
   IDENTIFIED BY 'password_b'
   ```
Random Password Generation

2. Wait for the password change to replicate throughout the system to all slave servers.

3. Modify each application that uses the `appuser1` account so that it connects to the servers using a password of `'password_b'` rather than `'password_a'`.

4. At this point, the secondary password is no longer needed. On each server that is not a replication slave, discard the secondary password:
   ```sql
   ALTER USER 'appuser1'@'host1.example.com'
   DISCARD OLD PASSWORD;
   ```

5. After the discard-password change has replicated to all slave servers, the credential change is complete.

The `RETAIN CURRENT PASSWORD` and `DISCARD OLD PASSWORD` clauses have the following effects:

- `RETAIN CURRENT PASSWORD` retains an account current password as its secondary password, replacing any existing secondary password. The new password becomes the primary password, but clients can use the account to connect to the server using either the primary or secondary password. (Exception: If the new password specified by the `ALTER USER` or `SET PASSWORD` statement is empty, the secondary password becomes empty as well, even if `RETAIN CURRENT PASSWORD` is given.)

- If you specify `RETAIN CURRENT PASSWORD` for an account that has an empty primary password, the statement fails.

- If an account has a secondary password and you change its primary password without specifying `RETAIN CURRENT PASSWORD`, the secondary password remains unchanged.

- For `ALTER USER`, if you change the authentication plugin assigned to the account, the secondary password is discarded. If you change the authentication plugin and also specify `RETAIN CURRENT PASSWORD`, the statement fails.

- For `ALTER USER`, `DISCARD OLD PASSWORD` discards the secondary password, if one exists. The account retains only its primary password, and clients can use the account to connect to the server only with the primary password.

Statements that modify secondary passwords require these privileges:

- The `APPLICATION_PASSWORD_ADMIN` privilege is required to use the `RETAIN CURRENT PASSWORD` or `DISCARD OLD PASSWORD` clause for `ALTER USER` and `SET PASSWORD` statements that apply to your own account. The privilege is required to manipulate your own secondary password because most users require only one password.

- If an account is to be permitted to manipulate secondary passwords for all accounts, it should be granted the `CREATE USER` privilege rather than `APPLICATION_PASSWORD_ADMIN`.

Random Password Generation

As of MySQL 8.0.18, the `CREATE USER`, `ALTER USER`, and `SET PASSWORD` statements have the capability of generating random passwords for user accounts, as an alternative to requiring explicit administrator-specified literal passwords. See the description of each statement for details about the syntax. This section describes the characteristics common to generated random passwords.

By default, generated random passwords have a length of 20 characters. This length is controlled by the `generated_random_password_length` system variable, which has a range from 5 to 255.

For each account for which a statement generates a random password, the statement stores the password in the `mysql.user` system table, hashed appropriately for the account authentication plugin. The statement also returns the cleartext password in a row of a result set to make it available to the user or application executing the statement. The result set columns are named `user`, `host`, and
Failed-Login Tracking and Temporary Account Locking

A CREATE USER, ALTER USER, or SET PASSWORD statement that generates a random password for an account is written to the binary log as a CREATE USER or ALTER USER statement with an IDENTIFIED WITH auth_plugin AS 'auth_string', clause, where auth_plugin is the account authentication plugin and 'auth_string' is the account hashed password value.

If the validate_password component is installed, the policy that it implements has no effect on generated passwords. (The purpose of password validation is to help humans create better passwords.)

Failed-Login Tracking and Temporary Account Locking

As of MySQL 8.0.19, administrators can configure user accounts such that too many consecutive login failures cause temporary account locking.

“Login failure” in this context means failure of the client to provide a correct password during a connection attempt. It does not include failure to connect for reasons such as unknown user or network issues. For accounts that have dual passwords (see Dual Password Support), either account password counts as correct.

The required number of login failures and the lock time are configurable per account, using the FAILED_LOGIN_ATTEMPTS and PASSWORD_LOCK_TIME options of the CREATE USER and ALTER USER statements. Examples:

CREATE USER 'u1'@'localhost' IDENTIFIED BY 'password'
    FAILED_LOGIN_ATTEMPTS 3 PASSWORD_LOCK_TIME 3;
ALTER USER 'u2'@'localhost'
    FAILED_LOGIN_ATTEMPTS 4 PASSWORD_LOCK_TIME UNBOUNDED;

When too many consecutive login failures occur, the client receives an error that looks like this:

ERROR 3957 (HY000): Access denied for user 'user'.
Account is blocked for $D day(s) ($R day(s) remaining)
due to $N consecutive failed logins.

Use the options as follows:

• FAILED_LOGIN_ATTEMPTS $N
This option indicates whether to track account login attempts that specify an incorrect password. The number \( N \) specifies how many consecutive incorrect passwords cause temporary account locking.

\[ \text{PASSWORD\_LOCK\_TIME} \{ N \mid \text{UNBOUNDED} \} \]

This option indicates how long to lock the account after too many consecutive login attempts provide an incorrect password. The value is a number \( N \) to specify the number of days the account remains locked, or \text{UNBOUNDED} to specify that when an account enters the temporarily locked state, the duration of that state is unbounded and does not end until the account is unlocked. The conditions under which unlocking occurs are described later.

Permitted values of \( N \) for each option are in the range from 0 to 32767. A value of 0 disables the option.

Failed-login tracking and temporary account locking have these characteristics:

- For failed-login tracking and temporary locking to occur for an account, its \text{FAILED\_LOGIN\_ATTEMPTS} and \text{PASSWORD\_LOCK\_TIME} options both must be nonzero.

- For \text{CREATE USER}, if \text{FAILED\_LOGIN\_ATTEMPTS} or \text{PASSWORD\_LOCK\_TIME} is not specified, its implicit default value is 0 for all accounts named by the statement. This means that failed-login tracking and temporary account locking are disabled. (These implicit defaults also apply to accounts created prior to the introduction of failed-login tracking.)

- For \text{ALTER USER}, if \text{FAILED\_LOGIN\_ATTEMPTS} or \text{PASSWORD\_LOCK\_TIME} is not specified, its value remains unchanged for all accounts named by the statement.

- For temporary account locking to occur, password failures must be consecutive. Any successful login that occurs prior to reaching the \text{FAILED\_LOGIN\_ATTEMPTS} value for failed logins causes failure counting to reset. For example, if \text{FAILED\_LOGIN\_ATTEMPTS} is 4 and three consecutive password failures have occurred, one more failure is necessary for locking to begin. But if the next login succeeds, failed-login counting for the account is reset so that four consecutive failures are again required for locking.

- Once temporary locking begins, successful login cannot occur even with the correct password until either the lock duration has passed or the account is unlocked by one of the account-reset methods listed in the following discussion.

When the server reads the grant tables, it initializes state information for each account regarding whether failed-login tracking is enabled, whether the account is currently temporarily locked and when locking began if so, and the number of failures before temporary locking occurs if the account is not locked.

An account’s state information can be reset, which means that failed-login counting is reset, and the account is unlocked if currently temporarily locked. Account resets can be global for all accounts or per account:

- A global reset of all accounts occurs for any of these conditions:
  - A server restart.
  - Execution of \text{FLUSH PRIVILEGES}. (Starting the server with \text{--skip-grant-tables} causes the grant tables not to be read, which disables failed-login tracking. In this case, the first execution of \text{FLUSH PRIVILEGES} causes the server to read the grant tables and enable failed-login tracking, in addition to resetting all accounts.)

- A per-account reset occurs for any of these conditions:
  - Successful login for the account.
  - The lock duration passes. In this case, failed-login counting resets at the time of the next login attempt.
- Execution of an `ALTER USER` statement for the account that sets either `FAILED_LOGIN_ATTEMPTS` or `PASSWORD_LOCK_TIME` (or both) to any value (including the current option value), or execution of an `ALTER USER ... UNLOCK` statement for the account.

Other `ALTER USER` statements for the account have no effect on its current failed-login count or its locking state.

Failed-login tracking is tied to the login account that is used to check credentials. If user proxying is in use, tracking occurs for the proxy user, not the proxied user. That is, tracking is tied to the account indicated by `USER()`, not the account indicated by `CURRENT_USER()`.

For information about the distinction between proxy and proxied users, see Section 4.18, "Proxy Users".

### 4.16 Server Handling of Expired Passwords

MySQL provides password-expiration capability, which enables database administrators to require that users reset their password. Passwords can be expired manually, and on the basis of a policy for automatic expiration (see Section 4.15, "Password Management").

The `ALTER USER` statement enables account password expiration. For example:

```
ALTER USER 'myuser'@'localhost' PASSWORD EXPIRE;
```

For each connection that uses an account with an expired password, the server either disconnects the client or restricts the client to “sandbox mode,” in which the server permits the client to perform only those operations necessary to reset the expired password. Which action is taken by the server depends on both client and server settings, as discussed later.

If the server disconnects the client, it returns an `ER_MUST_CHANGE_PASSWORD_LOGIN` error:

```
shell> mysql -u myuser -p
Password: ******
ERROR 1862 (HY000): Your password has expired. To log in you must change it using a client that supports expired passwords.
```

If the server restricts the client to sandbox mode, these operations are permitted within the client session:

- The client can reset the account password with `ALTER USER` or `SET PASSWORD`. After that has been done, the server restores normal access for the session, as well as for subsequent connections that use the account.

  **Note**

  Although it is possible to “reset” an expired password by setting it to its current value, it is preferable, as a matter of good policy, to choose a different password. DBAs can enforce non-reuse by establishing an appropriate password-reuse policy. See Password Reuse Policy.

- The client can use the `SET` statement.

For any operation not permitted within the session, the server returns an `ER_MUST_CHANGE_PASSWORD` error:

```
mysql> USE performance_schema;
ERROR 1820 (HY000): You must reset your password using ALTER USER statement before executing this statement.
mysql> SELECT 1;
ERROR 1820 (HY000): You must reset your password using ALTER USER statement before executing this statement.
```

That is what normally happens for interactive invocations of the `mysql` client because by default such invocations are put in sandbox mode. To resume normal functioning, select a new password.
For noninteractive invocations of the `mysql` client (for example, in batch mode), the server normally disconnects the client if the password is expired. To permit noninteractive `mysql` invocations to stay connected so that the password can be changed (using the statements permitted in sandbox mode), add the `--connect-expired-password` option to the `mysql` command.

As mentioned previously, whether the server disconnects an expired-password client or restricts it to sandbox mode depends on a combination of client and server settings. The following discussion describes the relevant settings and how they interact.

---

**Note**

This discussion applies only for accounts with expired passwords. If a client connects using a nonexpired password, the server handles the client normally.

---

On the client side, a given client indicates whether it can handle sandbox mode for expired passwords. For clients that use the C client library, there are two ways to do this:

- **Pass the** `MYSQL_OPT_CAN_HANDLE_EXPIRED_PASSWORDS` **flag to** `mysql_options()` **prior to connecting:**

  ```
  bool arg = 1;
  mysql_options(mysql,
                MYSQL_OPT_CAN_HANDLE_EXPIRED_PASSWORDS,
                &arg);
  ```

  This is the technique used within the `mysql` client, which enables `MYSQL_OPT_CAN_HANDLE_EXPIRED_PASSWORDS` if invoked interactively or with the `--connect-expired-password` option.

- **Pass the** `CLIENT_CAN_HANDLE_EXPIRED_PASSWORDS` **flag to** `mysql_real_connect()` **at connect time:**

  ```
  MYSQL mysql;
  mysql_init(&mysql);
  if (!mysql_real_connect(&mysql,
                          host, user, password, db,
                          port, unix_socket,
                          CLIENT_CAN_HANDLE_EXPIRED_PASSWORDS))
  { ... handle error ... }
  ```

Other MySQL Connectors have their own conventions for indicating readiness to handle sandbox mode. See the documentation for the Connector in which you are interested.

On the server side, if a client indicates that it can handle expired passwords, the server puts it in sandbox mode.

If a client does not indicate that it can handle expired passwords (or uses an older version of the client library that cannot so indicate), the server action depends on the value of the `disconnect_on_expired_password` system variable:

- **If** `disconnect_on_expired_password` **is enabled** (the default), the server disconnects the client with an `ER_MUST_CHANGE_PASSWORD_LOGIN` error.

- **If** `disconnect_on_expired_password` **is disabled**, the server puts the client in sandbox mode.

### 4.17 Pluggable Authentication

When a client connects to the MySQL server, the server uses the user name provided by the client and the client host to select the appropriate account row from the `mysql.user` system table. The server then authenticates the client, determining from the account row which authentication plugin applies to the client:
Available Authentication Plugins

• If the server cannot find the plugin, an error occurs and the connection attempt is rejected.

• Otherwise, the server invokes that plugin to authenticate the user, and the plugin returns a status to the server indicating whether the user provided the correct password and is permitted to connect.

Pluggable authentication enables these important capabilities:

• **Choice of authentication methods.** Pluggable authentication makes it easy for DBAs to choose and change the authentication method used for individual MySQL accounts.

• **External authentication.** Pluggable authentication makes it possible for clients to connect to the MySQL server with credentials appropriate for authentication methods that store credentials elsewhere than in the `mysql.user` system table. For example, plugins can be created to use external authentication methods such as PAM, Windows login IDs, LDAP, or Kerberos.

• **Proxy users:** If a user is permitted to connect, an authentication plugin can return to the server a user name different from the name of the connecting user, to indicate that the connecting user is a proxy for another user (the proxied user). While the connection lasts, the proxy user is treated, for purposes of access control, as having the privileges of the proxied user. In effect, one user impersonates another. For more information, see Section 4.18, “Proxy Users”.

**Note**

If you start the server with the `--skip-grant-tables` option, authentication plugins are not used even if loaded because the server performs no client authentication and permits any client to connect. Because this is insecure, if the server is started with the `--skip-grant-tables` option, it also disables remote connections by enabling `skip_networking`.

• Available Authentication Plugins
• Authentication Plugin Usage
• Authentication Plugin Client/Server Compatibility
• Authentication Plugin Connector-Writing Considerations
• Restrictions on Pluggable Authentication

**Available Authentication Plugins**

MySQL 8.0 provides these authentication plugins:

• A plugin that performs native authentication; that is, authentication based on the password hashing method in use from before the introduction of pluggable authentication in MySQL. The `mysql_native_password` plugin implements authentication based on this native password hashing method. See Section 6.1.1, “Native Pluggable Authentication”.

• Plugins that perform authentication using SHA-256 password hashing. This is stronger encryption than that available with native authentication. See Section 6.1.3, “SHA-256 Pluggable Authentication”, and Section 6.1.2, “Caching SHA-2 Pluggable Authentication”.

• A client-side plugin that sends the password to the server without hashing or encryption. This plugin is used in conjunction with server-side plugins that require access to the password exactly as provided by the client user. See Section 6.1.4, “Client-Side Cleartext Pluggable Authentication”.

• A plugin that performs external authentication using PAM (Pluggable Authentication Modules), enabling MySQL Server to use PAM to authenticate MySQL users. This plugin supports proxy users as well. See Section 6.1.5, “PAM Pluggable Authentication”.

• A plugin that performs external authentication on Windows, enabling MySQL Server to use native Windows services to authenticate client connections. Users who have logged in to Windows can connect from MySQL client programs to the server based on the information in their environment.
without specifying an additional password. This plugin supports proxy users as well. See Section 6.1.6, “Windows Pluggable Authentication”.

- Plugins that perform authentication using LDAP (Lightweight Directory Access Protocol) to authenticate MySQL users by accessing directory services such as X.500. These plugins support proxy users as well. See Section 6.1.7, “LDAP Pluggable Authentication”.

- A plugin that prevents all client connections to any account that uses it. Use cases for this plugin include proxied accounts that should never permit direct login but are accessed only through proxy accounts and accounts that must be able to execute stored programs and views with elevated privileges without exposing those privileges to ordinary users. See Section 6.1.8, “No-Login Pluggable Authentication”.

- A plugin that authenticates clients that connect from the local host through the Unix socket file. See Section 6.1.9, “Socket Peer-Credential Pluggable Authentication”.

- A test plugin that checks account credentials and logs success or failure to the server error log. This plugin is intended for testing and development purposes, and as an example of how to write an authentication plugin. See Section 6.1.10, “Test Pluggable Authentication”.

Note

For information about current restrictions on the use of pluggable authentication, including which connectors support which plugins, see Restrictions on Pluggable Authentication.

Third-party connector developers should read that section to determine the extent to which a connector can take advantage of pluggable authentication capabilities and what steps to take to become more compliant.

If you are interested in writing your own authentication plugins, see Writing Authentication Plugins.

Authentication Plugin Usage

This section provides general instructions for installing and using authentication plugins. For instructions specific to a given plugin, see the section that describes that plugin under Section 6.1, “Authentication Plugins”.

In general, pluggable authentication uses a pair of corresponding plugins on the server and client sides, so you use a given authentication method like this:

- If necessary, install the plugin library or libraries containing the appropriate plugins. On the server host, install the library containing the server-side plugin, so that the server can use it to authenticate client connections. Similarly, on each client host, install the library containing the client-side plugin for use by client programs. Authentication plugins that are built in need not be installed.

- For each MySQL account that you create, specify the appropriate server-side plugin to use for authentication. If the account is to use the default authentication plugin, the account-creation statement need not specify the plugin explicitly. The default_authentication_plugin system variable configures the default authentication plugin.

- When a client connects, the server-side plugin tells the client program which client-side plugin to use for authentication.

In the case that an account uses an authentication method that is the default for both the server and the client program, the server need not communicate to the client which client-side plugin to use, and a round trip in client/server negotiation can be avoided.

For standard MySQL clients such as mysql and mysqladmin, the --default-auth=plugin_name option can be specified on the command line as a hint about which client-side plugin the program can expect to use, although the server will override this if the server-side plugin associated with the user account requires a different client-side plugin.
If the client program does not find the client-side plugin library file, specify a `--plugin-dir=dir_name` option to indicate the plugin library directory location.

**Authentication Plugin Client/Server Compatibility**

Pluggable authentication enables flexibility in the choice of authentication methods for MySQL accounts, but in some cases client connections cannot be established due to authentication plugin incompatibility between the client and server.

The general compatibility principle for a successful client connection to a given account on a given server is that the client and server both must support the authentication *method* required by the account. Because authentication methods are implemented by authentication plugins, the client and server both must support the authentication *plugin* required by the account.

Authentication plugin incompatibilities can arise in various ways. Examples:

- Connect using a MySQL 5.7 client from 5.7.22 or lower to a MySQL 8.0 server account that authenticates with `caching_sha2_password`. This fails because the 5.7 client does not recognize the plugin, which was introduced in MySQL 8.0. (This issue is addressed in MySQL 5.7 as of 5.7.23, when `caching_sha2_password` client-side support was added to the MySQL client library and client programs.)

- Connect using a MySQL 5.5 client to a MySQL 5.6 server account that authenticates with `sha256_password`. This fails because the 5.5 client does not recognize the plugin, which was introduced in MySQL 5.6.

- Connect using a MySQL 5.7 client to a pre-5.7 server account that authenticates with `mysql_old_password`. This fails for multiple reasons. First, such a connection requires `--secure-auth=0`, which is no longer a supported option. Even were it supported, the 5.7 client does not recognize the plugin because it was removed in MySQL 5.7.

- Connect using a MySQL 5.7 client from a Community distribution to a MySQL 5.7 Enterprise server account that authenticates using one of the Enterprise-only LDAP authentication plugins. This fails because the Community client does not have access to the Enterprise plugin.

In general, these compatibility issues do not arise when connections are made between a client and server from the same MySQL distribution. When connections are made between a client and server from different MySQL series, issues can arise. These issues are inherent in the development process when MySQL introduces new authentication plugins or removes old ones. To minimize the potential for incompatibilities, regularly upgrade the server, clients, and connectors on a timely basis.

**Authentication Plugin Connector-Writing Considerations**

Various implementations of the MySQL client/server protocol exist. The `libmysqlclient` C API client library is one implementation. Some MySQL connectors (typically those not written in C) provide their own implementation. However, not all protocol implementations handle plugin authentication the same way. This section describes an authentication issue that protocol implementors should take into account.

In the client/server protocol, the server tells connecting clients which authentication plugin it considers the default. If the protocol implementation used by the client tries to load the default plugin and that plugin does not exist on the client side, the load operation fails. This is an unnecessary failure if the default plugin is not the plugin actually required by the account to which the client is trying to connect.

If a client/server protocol implementation does not have its own notion of default authentication plugin and always tries to load the default plugin specified by the server, it will fail with an error if that plugin is not available.

To avoid this problem, the protocol implementation used by the client should have its own default plugin and should use it as its first choice (or, alternatively, fall back to this default in case of failure to load the default plugin specified by the server). Example:
Restrictions on Pluggable Authentication

The first part of this section describes general restrictions on the applicability of the pluggable authentication framework described at Section 4.17, “Pluggable Authentication”. The second part describes how third-party connector developers can determine the extent to which a connector can take advantage of pluggable authentication capabilities and what steps to take to become more compliant.

The term “native authentication” used here refers to authentication against passwords stored in the `mysql.user` system table. This is the same authentication method provided by older MySQL servers, before pluggable authentication was implemented. “Windows native authentication” refers to authentication using the credentials of a user who has already logged in to Windows, as implemented by the Windows Native Authentication plugin (“Windows plugin” for short).

- **General Pluggable Authentication Restrictions**
- **Pluggable Authentication and Third-Party Connectors**

**General Pluggable Authentication Restrictions**

- **Connector/C++**: Clients that use this connector can connect to the server only through accounts that use native authentication.

  Exception: A connector supports pluggable authentication if it was built to link to `libmysqlclient` dynamically (rather than statically) and it loads the current version of `libmysqlclient` if that version is installed, or if the connector is recompiled from source to link against the current `libmysqlclient`.

  For information about writing connectors to handle information from the server about the default server-side authentication plugin, see Authentication Plugin Connector-Writing Considerations.

- **Connector/NET**: Clients that use Connector/NET can connect to the server through accounts that use native authentication or Windows native authentication.

- **Connector/PHP**: Clients that use this connector can connect to the server only through accounts that use native authentication, when compiled using the MySQL native driver for PHP (`mysqlnd`).

- **Windows native authentication**: Connecting through an account that uses the Windows plugin requires Windows Domain setup. Without it, NTLM authentication is used and then only local connections are possible; that is, the client and server must run on the same computer.

- **Proxy users**: Proxy user support is available to the extent that clients can connect through accounts authenticated with plugins that implement proxy user capability (that is, plugins that can return a user name different from that of the connecting user). For example, the PAM and Windows plugins support proxy users. The `mysql_native_password` and `sha256_password` authentication plugins do not support proxy users by default, but can be configured to do so; see Server Support for Proxy User Mapping.

- **Replication**: Replication slaves can employ not only master accounts using native authentication, but can also connect through master accounts that use nonnative authentication if the required
client-side plugin is available. If the plugin is built into `libmysqlclient`, it is available by default. Otherwise, the plugin must be installed on the slave side in the directory named by the slave `plugin_dir` system variable.

- **FEDERATED tables**: A FEDERATED table can access the remote table only through accounts on the remote server that use native authentication.

**Pluggable Authentication and Third-Party Connectors**

Third-party connector developers can use the following guidelines to determine readiness of a connector to take advantage of pluggable authentication capabilities and what steps to take to become more compliant:

- An existing connector to which no changes have been made uses native authentication and clients that use the connector can connect to the server only through accounts that use native authentication. However, you should test the connector against a recent version of the server to verify that such connections still work without problem.

  Exception: A connector might work with pluggable authentication without any changes if it links to `libmysqlclient` dynamically (rather than statically) and it loads the current version of `libmysqlclient` if that version is installed.

- To take advantage of pluggable authentication capabilities, a connector that is `libmysqlclient`-based should be relinked against the current version of `libmysqlclient`. This enables the connector to support connections through accounts that require client-side plugins now built into `libmysqlclient` (such as the cleartext plugin needed for PAM authentication and the Windows plugin needed for Windows native authentication). Linking with a current `libmysqlclient` also enables the connector to access client-side plugins installed in the default MySQL plugin directory (typically the directory named by the default value of the local server's `plugin_dir` system variable).

  If a connector links to `libmysqlclient` dynamically, it must be ensured that the newer version of `libmysqlclient` is installed on the client host and that the connector loads it at runtime.

- Another way for a connector to support a given authentication method is to implement it directly in the client/server protocol. Connector/NET uses this approach to provide support for Windows native authentication.

- If a connector should be able to load client-side plugins from a directory different from the default plugin directory, it must implement some means for client users to specify the directory. Possibilities for this include a command-line option or environment variable from which the connector can obtain the directory name. Standard MySQL client programs such as `mysql` and `mysqladmin` implement a `--plugin-dir` option. See also [C API Client Plugin Functions](#).

- Proxy user support by a connector depends, as described earlier in this section, on whether the authentication methods that it supports permit proxy users.

### 4.18 Proxy Users

The MySQL server authenticates client connections using authentication plugins. The plugin that authenticates a given connection may request that the connecting (external) user be treated as a different user for privilege-checking purposes. This enables the external user to be a proxy for the second user; that is, to assume the privileges of the second user:

- The external user is a “proxy user” (a user who can impersonate or become known as another user).

- The second user is a “proxied user” (a user whose identity and privileges can be assumed by a proxy user).

This section describes how the proxy user capability works. For general information about authentication plugins, see [Section 4.17, “Pluggable Authentication”](#). For information about specific
plugins, see Section 6.1, “Authentication Plugins”. For information about writing authentication plugins that support proxy users, see Implementing Proxy User Support in Authentication Plugins.

- Requirements for Proxy User Support
- Simple Proxy User Example
- Preventing Direct Login to Proxied Accounts
- Granting and Revoking the PROXY Privilege
- Default Proxy Users
- Default Proxy User and Anonymous User Conflicts
- Server Support for Proxy User Mapping
- Proxy User System Variables

**Note**

One administrative benefit to be gained by proxying is that the DBA can set up a single account with a set of privileges and then enable multiple proxy users to have those privileges without having to assign the privileges individually to each of those users. As an alternative to proxy users, DBAs may find that roles provide a suitable way to map users onto specific sets of named privileges. Each user can be granted a given single role to, in effect, be granted the appropriate set of privileges. See Section 4.10, “Using Roles”.

**Requirements for Proxy User Support**

For proxying to occur for a given authentication plugin, these conditions must be satisfied:

- Proxying must be supported, either by the plugin itself, or by the MySQL server on behalf of the plugin. In the latter case, server support may need to be enabled explicitly; see Server Support for Proxy User Mapping.

- The account for the external proxy user must be set up to be authenticated by the plugin. Use the CREATE USER statement to associate an account with an authentication plugin, or ALTER USER to change its plugin.

- The account for the proxied user must exist and be granted the privileges to be assumed by the proxy user. Use the CREATE USER and GRANT statements for this.

- Normally, the proxied user is configured so that it can be used only in proxying scenarios and not for direct logins.

- The proxy user account must have the PROXY privilege for the proxied account. Use the GRANT statement for this.

- For a client connecting to the proxy account to be treated as a proxy user, the authentication plugin must return a user name different from the client user name, to indicate the user name of the proxied account that defines the privileges to be assumed by the proxy user.

  Alternatively, for plugins that are provided proxy mapping by the server, the proxied user is determined from the PROXY privilege held by the proxy user.

  The proxy mechanism permits mapping only the external client user name to the proxied user name. There is no provision for mapping host names:

  - When a client connects to the server, the server determines the proper account based on the user name passed by the client program and the host from which the client connects.
If that account is a proxy account, the server attempts to determine the appropriate proxied account by finding a match for a proxied account using the user name returned by the authentication plugin and the host name of the proxy account. The host name in the proxied account is ignored.

**Simple Proxy User Example**

Consider the following account definitions:

```
-- create proxy account
CREATE USER 'employee_ext'@'localhost'
  IDENTIFIED WITH my_auth_plugin
  AS 'my_auth_string';

-- create proxied account and grant its privileges;
-- use mysql_no_login plugin to prevent direct login
CREATE USER 'employee'@'localhost'
  IDENTIFIED WITH mysql_no_login;
GRANT ALL
  ON employees.*
  TO 'employee'@'localhost';
GRANT PROXY
  ON 'employee'@'localhost'
  TO 'employee_ext'@'localhost';
```

When a client connects as `employee_ext` from the local host, MySQL uses the plugin named `my_auth_plugin` to perform authentication. Suppose that `my_auth_plugin` returns a user name of `employee` to the server, based on the content of `my_auth_string` and perhaps by consulting some external authentication system. The name `employee` differs from `employee_ext`, so returning `employee` serves as a request to the server to treat the `employee_ext` external user, for purposes of privilege checking, as the `employee` local user.

In this case, `employee_ext` is the proxy user and `employee` is the proxied user.

The server verifies that proxy authentication for `employee` is possible for the `employee_ext` user by checking whether `employee_ext` (the proxy user) has the `PROXY` privilege for `employee` (the proxied user). If this privilege has not been granted, an error occurs. Otherwise, `employee_ext` assumes the privileges of `employee`. The server checks statements executed during the client session by `employee_ext` against the privileges granted to `employee`. In this case, `employee_ext` can access tables in the `employees` database.

The proxied account, `employee`, uses the `mysql_no_login` authentication plugin to prevent clients from using the account to log in directly. (This assumes that the plugin is installed. For instructions, see Section 6.1.8, “No-Login Pluggable Authentication”.) For alternative methods of protecting proxied accounts against direct use, see Preventing Direct Login to Proxied Accounts.

When proxying occurs, the `USER()` and `CURRENT_USER()` functions can be used to see the difference between the connecting user (the proxy user) and the account whose privileges apply during the current session (the proxied user). For the example just described, those functions return these values:

```
mysql> SELECT USER(), CURRENT_USER();
+------------------------+--------------------+
| USER()                 | CURRENT_USER()     |
|------------------------+--------------------|
| employee_ext@localhost | employee@localhost |
+------------------------+--------------------+
```

In the `CREATE USER` statement that creates the proxy user account, the `IDENTIFIED WITH` clause that names the proxy-supporting authentication plugin is optionally followed by an `AS 'auth_string'` clause specifying a string that the server passes to the plugin when the user connects. If present, the string provides information that helps the plugin determine how to map the proxy (external) client user name to a proxied user name. It is up to each plugin whether it requires the `AS` clause. If so, the format of the authentication string depends on how the plugin intends to use it. Consult the documentation for a given plugin for information about the authentication string values it accepts.
**Preventing Direct Login to Proxied Accounts**

Proxied accounts generally are intended to be used only by means of proxy accounts. That is, clients connect using a proxy account, then are mapped onto and assume the privileges of the appropriate proxied user.

There are multiple ways to ensure that a proxied account cannot be used directly:

- Associate the account with the `mysql_no_login` authentication plugin. In this case, the account cannot be used for direct logins under any circumstances. This assumes that the plugin is installed. For instructions, see Section 6.1.8, "No-Login Pluggable Authentication".

- Include the `ACCOUNT LOCK` option when you create the account. See CREATE USER Statement. With this method, also include a password so that if the account is unlocked later, it cannot be accessed with no password. (If the `validate_password` component is enabled, it will not permit creating an account without a password, even if the account is locked. See Section 6.3, "The Password Validation Component".)

- Create the account with a password but do not tell anyone else the password. If you do not let anyone know the password for the account, clients cannot use it to connect directly to the MySQL server.

**Granting and Revoking the PROXY Privilege**

The `PROXY` privilege is needed to enable an external user to connect as and have the privileges of another user. To grant this privilege, use the `GRANT` statement. For example:

```
GRANT PROXY ON 'proxied_user' TO 'proxy_user';
```

The statement creates a row in the `mysql.proxies_priv` grant table.

At connect time, `proxy_user` must represent a valid externally authenticated MySQL user, and `proxied_user` must represent a valid locally authenticated user. Otherwise, the connection attempt fails.

The corresponding `REVOKE` syntax is:

```
REVOKE PROXY ON 'proxied_user' FROM 'proxy_user';
```

MySQL `GRANT` and `REVOKE` syntax extensions work as usual. Examples:

```
-- grant PROXY to multiple accounts
GRANT PROXY ON 'a' TO 'b', 'c', 'd';
-- revoke PROXY from multiple accounts
REVOKE PROXY ON 'a' FROM 'b', 'c', 'd';
-- grant PROXY to an account and enable the account to grant
-- PROXY to the proxied account
GRANT PROXY ON 'a' TO 'd' WITH GRANT OPTION;
-- grant PROXY to default proxy account
GRANT PROXY ON 'a' TO '@';
```

The `PROXY` privilege can be granted in these cases:

- By a user that has `GRANT PROXY ... WITH GRANT OPTION` for `proxied_user`.

- By `proxied_user` for itself: The value of `USER()` must exactly match `CURRENT_USER()` and `proxied_user`, for both the user name and host name parts of the account name.

The initial `root` account created during MySQL installation has the `PROXY ... WITH GRANT OPTION` privilege for `''@'`, that is, for all users and all hosts. This enables `root` to set up proxy users, as well as to delegate to other accounts the authority to set up proxy users. For example, `root` can do this:

```
CREATE USER 'admin'@'localhost'
```
Those statements create an `admin` user that can manage all `GRANT PROXY` mappings. For example, `admin` can do this:

```
GRANT PROXY ON sally TO joe;
```

### Default Proxy Users

To specify that some or all users should connect using a given authentication plugin, create a “blank” MySQL account with an empty user name and host name (`''@''`), associate it with that plugin, and let the plugin return the real authenticated user name (if different from the blank user). Suppose that there exists a plugin named `ldap_auth` that implements LDAP authentication and maps connecting users onto either a developer or manager account. To set up proxying of users onto these accounts, use the following statements:

```sql
-- create default proxy account
CREATE USER ''@''
IDENTIFIED WITH ldap_auth
AS 'O=Oracle, OU=MySQL';
-- create proxied accounts; use mysql_no_login plugin to prevent direct login
CREATE USER 'developer'@'localhost'
IDENTIFIED WITH mysql_no_login;
CREATE USER 'manager'@'localhost'
IDENTIFIED WITH mysql_no_login;
-- grant to default proxy account the PROXY privilege for proxied accounts
GRANT PROXY
ON 'manager'@'localhost'
TO ''@'';
GRANT PROXY
ON 'developer'@'localhost'
TO ''@'';
```

Now assume that a client connects as follows:

```
shell> mysql --user=myuser --password ... Enter password: myuser_password
```

The server will not find `myuser` defined as a MySQL user. But because there is a blank user account (`''@''`) that matches the client user name and host name, the server authenticates the client against that account: The server invokes the `ldap_auth` authentication plugin and passes `myuser` and `myuser_password` to it as the user name and password.

If the `ldap_auth` plugin finds in the LDAP directory that `myuser_password` is not the correct password for `myuser`, authentication fails and the server rejects the connection.

If the password is correct and `ldap_auth` finds that `myuser` is a developer, it returns the user name `developer` to the MySQL server, rather than `myuser`. Returning a user name different from the client user name of `myuser` signals to the server that it should treat `myuser` as a proxy. The server verifies that `''@''` can authenticate as `developer` (because `''@''` has the `PROXY` privilege to do so) and accepts the connection. The session proceeds with `myuser` having the privileges of the `developer` proxied user. (These privileges should be set up by the DBA using `GRANT` statements, not shown.)

The `USER()` and `CURRENT_USER()` functions return these values:

```
mysql> SELECT USER(), CURRENT_USER();
```

```
+------------------+---------------------+
| USER()           | CURRENT_USER()      |
|------------------+---------------------+
| myuser@localhost | developer@localhost |
```

```
Default Proxy User and Anonymous User Conflicts

If the plugin instead finds in the LDAP directory that myuser is a manager, it returns manager as the user name and the session proceeds with myuser having the privileges of the manager proxied user.

```
mysql> SELECT USER(), CURRENT_USER();
+------------------+-------------------+
| USER()           | CURRENT_USER()    |
+------------------+-------------------+
| myuser@localhost | manager@localhost |
+------------------+-------------------+
```

For simplicity, external authentication cannot be multilevel: Neither the credentials for developer nor those for manager are taken into account in the preceding example. However, they are still used if a client tries to connect and authenticate directly as the developer or manager account, which is why those proxied accounts should be protected against direct login (see Preventing Direct Login to Proxied Accounts).

**Default Proxy User and Anonymous User Conflicts**

If you intend to create a default proxy user, check for other existing “match any user” accounts that take precedence over the default proxy user because they can prevent that user from working as intended.

In the preceding discussion, the default proxy user account has ‘’ in the host part, which matches any host. If you set up a default proxy user, take care to also check whether nonproxy accounts exist with the same user part and ‘%’ in the host part, because ‘%’ also matches any host, but has precedence over ‘’ by the rules that the server uses to sort account rows internally (see Section 4.6, “Access Control, Stage 1: Connection Verification”).

Suppose that a MySQL installation includes these two accounts:

```
-- create default proxy account
CREATE USER ''@''
  IDENTIFIED WITH some_plugin
  AS 'some_auth_string';
-- create anonymous account
CREATE USER ''@%'
  IDENTIFIED BY 'anon_user_password';
```

The first account (’’@’’) is intended as the default proxy user, used to authenticate connections for users who do not otherwise match a more-specific account. The second account (’’@%’’) is an anonymous-user account, which might have been created, for example, to enable users without their own account to connect anonymously.

Both accounts have the same user part (’’), which matches any user. And each account has a host part that matches any host. Nevertheless, there is a priority in account matching for connection attempts because the matching rules sort a host of ‘%’ ahead of ‘’. For accounts that do not match any more-specific account, the server attempts to authenticate them against ’’@%’’ (the anonymous user) rather than ’’@’’ (the default proxy user). As a result, the default proxy account is never used.

To avoid this problem, use one of the following strategies:

- Remove the anonymous account so that it does not conflict with the default proxy user.
- Use a more-specific default proxy user that matches ahead of the anonymous user. For example, to permit only localhost proxy connections, use ’’@localhost’’:

```
CREATE USER ''@localhost'
  IDENTIFIED WITH some_plugin
  AS 'some_auth_string';
```

In addition, modify any GRANT PROXY statements to name ’’@localhost’’ rather than ’’@’’ as the proxy user.

Be aware that this strategy prevents anonymous-user connections from localhost.
• Use a named default account rather than an anonymous default account. For an example of this technique, consult the instructions for using the `authentication_windows` plugin. See Section 6.1.6, “Windows Pluggable Authentication”.

• Create multiple proxy users, one for local connections and one for “everything else” (remote connections). This can be useful particularly when local users should have different privileges from remote users.

Create the proxy users:

```
-- create proxy user for local connections
CREATE USER ''@'localhost'
    IDENTIFIED WITH some_plugin
    AS 'some_auth_string';

-- create proxy user for remote connections
CREATE USER ''@'%'
    IDENTIFIED WITH some_plugin
    AS 'some_auth_string';
```

Create the proxied users:

```
-- create proxied user for local connections
CREATE USER 'developer'@'localhost'
    IDENTIFIED WITH mysql_no_login;

-- create proxied user for remote connections
CREATE USER 'developer'@'%'
    IDENTIFIED WITH mysql_no_login;
```

Grant to each proxy account the `PROXY` privilege for the corresponding proxied account:

```
GRANT PROXY
    ON 'developer'@'localhost'
    TO ''@'localhost';

GRANT PROXY
    ON 'developer'@'%'
    TO ''@'%';
```

Finally, grant appropriate privileges to the local and remote proxied users (not shown).

Assume that the `some_plugin/some_auth_string` combination causes `some_plugin` to map the client user name to `developer`. Local connections match the `'@'localhost` proxy user, which maps to the `'developer'@'localhost` proxied user. Remote connections match the `'@'%` proxy user, which maps to the `'developer'@'%` proxied user.

### Server Support for Proxy User Mapping

Some authentication plugins implement proxy user mapping for themselves (for example, the PAM and Windows authentication plugins). Other authentication plugins do not support proxy users by default. Of these, some can request that the MySQL server itself map proxy users according to granted proxy privileges: `mysql_native_password`, `sha256_password`. If the `check_proxy_users` system variable is enabled, the server performs proxy user mapping for any authentication plugins that make such a request:

• By default, `check_proxy_users` is disabled, so the server performs no proxy user mapping even for authentication plugins that request server support for proxy users.

• If `check_proxy_users` is enabled, it may also be necessary to enable a plugin-specific system variable to take advantage of server proxy user mapping support:

  • For the `mysql_native_password` plugin, enable `mysql_native_password_proxy_users`.
  
  • For the `sha256_password` plugin, enable `sha256_password_proxy_users`.

For example, to enable all the preceding capabilities, start the server with these lines in the `my.cnf` file:
Proxy User System Variables

Assuming that the relevant system variables have been enabled, create the proxy user as usual using `CREATE USER`, then grant it the `PROXY` privilege to a single other account to be treated as the proxied user. When the server receives a successful connection request for the proxy user, it finds that the user has the `PROXY` privilege and uses it to determine the proper proxied user.

```sql
-- create proxy account
CREATE USER 'proxy_user'@'localhost'
  IDENTIFIED WITH mysql_native_password
  BY 'password';
-- create proxied account and grant its privileges;
-- use mysql_no_login plugin to prevent direct login
CREATE USER 'proxied_user'@'localhost'
  IDENTIFIED WITH mysql_no_login;
-- grant privileges to proxied account
GRANT ...
  ON ...
  TO 'proxied_user'@'localhost';
-- grant to proxy account the
-- PROXY privilege for proxied account
GRANT PROXY
  ON 'proxied_user'@'localhost'
  TO 'proxy_user'@'localhost';
```

To use the proxy account, connect to the server using its name and password:

```
shell> mysql -u proxy_user -p
Enter password: (enter proxy_user password here)
```

Authentication succeeds, the server finds that `proxy_user` has the `PROXY` privilege for `proxied_user`, and the session proceeds with `proxy_user` having the privileges of `proxied_user`.

Proxy user mapping performed by the server is subject to these restrictions:

- The server will not proxy to or from an anonymous user, even if the associated `PROXY` privilege is granted.
- When a single account has been granted proxy privileges for more than one proxied account, server proxy user mapping is nondeterministic. Therefore, granting to a single account proxy privileges for multiple proxied accounts is discouraged.

**Proxy User System Variables**

Two system variables help trace the proxy login process:

- `proxy_user`: This value is `NULL` if proxying is not used. Otherwise, it indicates the proxy user account. For example, if a client authenticates through the `'@'` proxy account, this variable is set as follows:

  ```sql
  mysql> SELECT @@proxy_user;
  +---------------+
  | @proxy_user   |
  +---------------+
  | '@'           |
  +---------------+
  ```

- `external_user`: Sometimes the authentication plugin may use an external user to authenticate to the MySQL server. For example, when using Windows native authentication, a plugin that authenticates using the windows API does not need the login ID passed to it. However, it still uses a Windows user ID to authenticate. The plugin may return this external user ID (or the first 512 UTF-8 bytes of it) to the server using the `external_user` read-only session variable. If the plugin does not set this variable, its value is `NULL`.
4.19 Account Locking

MySQL supports locking and unlocking user accounts using the `ACCOUNT LOCK` and `ACCOUNT UNLOCK` clauses for the `CREATE USER` and `ALTER USER` statements:

- When used with `CREATE USER`, these clauses specify the initial locking state for a new account. In the absence of either clause, the account is created in an unlocked state.

  If the `validate_password` component is enabled, it will not permit creating an account without a password, even if the account is locked. See Section 6.3, “The Password Validation Component”.

- When used with `ALTER USER`, these clauses specify the new locking state for an existing account. In the absence of either clause, the account locking state remains unchanged.

As of MySQL 8.0.19, `ALTER USER ... UNLOCK` unlocks any account named by the statement that is temporarily locked due to too many failed logins. See Section 4.15, “Password Management”.

Account locking state is recorded in the `account_locked` column of the `mysql.user` system table. The output from `SHOW CREATE USER` indicates whether an account is locked or unlocked.

If a client attempts to connect to a locked account, the attempt fails. The server increments the `Locked_connects` status variable that indicates the number of attempts to connect to a locked account, returns an `ER_ACCOUNT_HAS_BEEN_LOCKED` error, and writes a message to the error log:

```
Access denied for user 'user_name'@'host_name'.
Account is locked.
```

Locking an account does not affect being able to connect using a proxy user that assumes the identity of the locked account. It also does not affect the ability to execute stored programs or views that have a `DEFINER` attribute naming the locked account. That is, the ability to use a proxied account or stored programs or views is not affected by locking the account.

The account-locking capability depends on the presence of the `account_locked` column in the `mysql.user` system table. For upgrades from MySQL versions older than 5.7.6, perform the MySQL upgrade procedure to ensure that this column exists. See Upgrading MySQL. For nonupgraded installations that have no `account_locked` column, the server treats all accounts as unlocked, and using the `ACCOUNT LOCK` or `ACCOUNT UNLOCK` clauses produces an error.

4.20 Setting Account Resource Limits

One means of restricting client use of MySQL server resources is to set the global `max_user_connections` system variable to a nonzero value. This limits the number of simultaneous connections that can be made by any given account, but places no limits on what a client can do once connected. In addition, setting `max_user_connections` does not enable management of individual accounts. Both types of control are of interest to MySQL administrators.

To address such concerns, MySQL permits limits for individual accounts on use of these server resources:

- The number of queries an account can issue per hour
- The number of updates an account can issue per hour
- The number of times an account can connect to the server per hour
- The number of simultaneous connections to the server by an account

Any statement that a client can issue counts against the query limit. Only statements that modify databases or tables count against the update limit.

An “account” in this context corresponds to a row in the `mysql.user` system table. That is, a connection is assessed against the `User` and `Host` values in the `user` table row that applies to the
connection. For example, an account 'usera'@'%.example.com' corresponds to a row in the user table that has User and Host values of usera and %.example.com, to permit usera to connect from any host in the example.com domain. In this case, the server applies resource limits in this row collectively to all connections by usera from any host in the example.com domain because all such connections use the same account.

Before MySQL 5.0, an "account" was assessed against the actual host from which a user connects. This older method of accounting may be selected by starting the server with the --old-style-user-limits option. In this case, if usera connects simultaneously from host1.example.com and host2.example.com, the server applies the account resource limits separately to each connection. If usera connects again from host1.example.com, the server applies the limits for that connection together with the existing connection from that host.

To establish resource limits for an account at account-creation time, use the CREATE USER statement. To modify the limits for an existing account, use ALTER USER. Provide a WITH clause that names each resource to be limited. The default value for each limit is zero (no limit). For example, to create a new account that can access the customer database, but only in a limited fashion, issue these statements:

```
mysql> CREATE USER 'francis'@'localhost' IDENTIFIED BY 'frank'
    -> WITH MAX_QUERIES_PER_HOUR 20
    -> MAX_UPDATES_PER_HOUR 10
    -> MAX_CONNECTIONS_PER_HOUR 5
    -> MAX_USER_CONNECTIONS 2;
```

The limit types need not all be named in the WITH clause, but those named can be present in any order. The value for each per-hour limit should be an integer representing a count per hour. For MAX_USER_CONNECTIONS, the limit is an integer representing the maximum number of simultaneous connections by the account. If this limit is set to zero, the global max_user_connections system variable value determines the number of simultaneous connections. If max_user_connections is also zero, there is no limit for the account.

To modify limits for an existing account, use an ALTER USER statement. The following statement changes the query limit for francis to 100:

```
mysql> ALTER USER 'francis'@'localhost' WITH MAX_QUERIES_PER_HOUR 100;
```

The statement modifies only the limit value specified and leaves the account otherwise unchanged.

To remove a limit, set its value to zero. For example, to remove the limit on how many times per hour francis can connect, use this statement:

```
mysql> ALTER USER 'francis'@'localhost' WITH MAX_CONNECTIONS_PER_HOUR 0;
```

As mentioned previously, the simultaneous-connection limit for an account is determined from the MAX_USER_CONNECTIONS limit and the max_user_connections system variable. Suppose that the global max_user_connections value is 10 and three accounts have individual resource limits specified as follows:

```
ALTER USER 'user1'@'localhost' WITH MAX_USER_CONNECTIONS 0;
ALTER USER 'user2'@'localhost' WITH MAX_USER_CONNECTIONS 5;
ALTER USER 'user3'@'localhost' WITH MAX_USER_CONNECTIONS 20;
```

user1 has a connection limit of 10 (the global max_user_connections value) because it has a MAX_USER_CONNECTIONS limit of zero. user2 and user3 have connection limits of 5 and 20, respectively, because they have nonzero MAX_USER_CONNECTIONS limits.

The server stores resource limits for an account in the user table row corresponding to the account. The max_questions, max_updates, and max_connections columns store the per-hour limits, and the max_user_connections column stores the MAX_USER_CONNECTIONS limit. (See Section 4.3, "Grant Tables").

Resource-use counting takes place when any account has a nonzero limit placed on its use of any of the resources.
As the server runs, it counts the number of times each account uses resources. If an account reaches its limit on number of connections within the last hour, the server rejects further connections for the account until that hour is up. Similarly, if the account reaches its limit on the number of queries or updates, the server rejects further queries or updates until the hour is up. In all such cases, the server issues appropriate error messages.

Resource counting occurs per account, not per client. For example, if your account has a query limit of 50, you cannot increase your limit to 100 by making two simultaneous client connections to the server. Queries issued on both connections are counted together.

The current per-hour resource-use counts can be reset globally for all accounts, or individually for a given account:

- To reset the current counts to zero for all accounts, issue a `FLUSH USER_RESOURCES` statement. The counts also can be reset by reloading the grant tables (for example, with a `FLUSH PRIVILEGES` statement or a `mysqladmin reload` command).
- The counts for an individual account can be reset to zero by setting any of its limits again. Specify a limit value equal to the value currently assigned to the account.

Per-hour counter resets do not affect the `MAX_USER_CONNECTIONS` limit.

All counts begin at zero when the server starts. Counts do not carry over through server restarts.

For the `MAX_USER_CONNECTIONS` limit, an edge case can occur if the account currently has open the maximum number of connections permitted to it: A disconnect followed quickly by a connect can result in an error (`ER_TOO_MANY_USER_CONNECTIONS` or `ER_USER_LIMIT_REACHED`) if the server has not fully processed the disconnect by the time the connect occurs. When the server finishes disconnect processing, another connection will once more be permitted.

### 4.21 Troubleshooting Problems Connecting to MySQL

If you encounter problems when you try to connect to the MySQL server, the following items describe some courses of action you can take to correct the problem.

- Make sure that the server is running. If it is not, clients cannot connect to it. For example, if an attempt to connect to the server fails with a message such as one of those following, one cause might be that the server is not running:

```shell
code
shell> mysql
ERROR 2003: Can't connect to MySQL server on 'host_name' (111)
```

- It might be that the server is running, but you are trying to connect using a TCP/IP port, named pipe, or Unix socket file different from the one on which the server is listening. To correct this when you invoke a client program, specify a `--port` option to indicate the proper port number, or a `--socket` option to indicate the proper named pipe or Unix socket file. To find out where the socket file is, you can use this command:

```shell
code
shell> netstat -ln | grep mysql
```

- Make sure that the server has not been configured to ignore network connections or (if you are attempting to connect remotely) that it has not been configured to listen only locally on its network interfaces. If the server was started with the `skip_networking` system variable enabled, it will not accept TCP/IP connections at all. If the server was started with the `bind_address` system variable set to `127.0.0.1`, it will listen for TCP/IP connections only locally on the loopback interface and will not accept remote connections.

- Check to make sure that there is no firewall blocking access to MySQL. Your firewall may be configured on the basis of the application being executed, or the port number used by MySQL for
communication (3306 by default). Under Linux or Unix, check your IP tables (or similar) configuration
to ensure that the port has not been blocked. Under Windows, applications such as ZoneAlarm or
Windows Firewall may need to be configured not to block the MySQL port.

• The grant tables must be properly set up so that the server can use them for access control. For
some distribution types (such as binary distributions on Windows, or RPM distributions on Linux),
the installation process initializes the MySQL data directory, including the mysql system database
containing the grant tables. For distributions that do not do this, you must initialize the data directory
manually. For details, see Chapter 3, Postinstallation Setup and Testing.

To determine whether you need to initialize the grant tables, look for a mysql directory under the
data directory. (The data directory normally is named data or var and is located under your MySQL
installation directory.) Make sure that you have a file named user.MYD in the mysql database
directory. If not, initialize the data directory. After doing so and starting the server, you should be able
to connect to the server.

• After a fresh installation, if you try to log on to the server as root without using a password, you
might get the following error message.

    shell> mysql -u root
    ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: NO)

It means a root password has already been assigned during installation and it has to be supplied.
See Section 3.4, “Securing the Initial MySQL Account” on the different ways the password could
have been assigned and, in some cases, how to find it. If you need to reset the root password, see
instructions in How to Reset the Root Password. After you have found or reset your password, log on
again as root using the --password (or -p) option:

    shell> mysql -u root -p
    Enter password:

However, the server is going to let you connect as root without using a password if you have
initialized MySQL using mysqld --initialize-insecure (see Section 3.1, “Initializing the Data
Directory” for details). That is a security risk, so you should set a password for the root account; see
Section 3.4, “Securing the Initial MySQL Account” for instructions.

• If you have updated an existing MySQL installation to a newer version, did you perform the MySQL
upgrade procedure? If not, do so. The structure of the grant tables changes occasionally when new
capabilities are added, so after an upgrade you should always make sure that your tables have the
current structure. For instructions, see Upgrading MySQL.

• If a client program receives the following error message when it tries to connect, it means that the
server expects passwords in a newer format than the client is capable of generating:

    shell> mysql
    Client does not support authentication protocol requested
    by server; consider upgrading MySQL client

• Remember that client programs use connection parameters specified in option files or
environment variables. If a client program seems to be sending incorrect default connection
parameters when you have not specified them on the command line, check any applicable option
files and your environment. For example, if you get Access denied when you run a client without
any options, make sure that you have not specified an old password in any of your option files!

You can suppress the use of option files by a client program by invoking it with the --no-defaults
option. For example:

    shell> mysqladmin --no-defaults -u root version

The option files that clients use are listed in Using Option Files. Environment variables are listed in
Environment Variables.

• If you get the following error, it means that you are using an incorrect root password:
Troubleshooting Problems Connecting to MySQL

shell> mysqladmin -u root -pxxxx ver
Access denied for user 'root'@'localhost' (using password: YES)

If the preceding error occurs even when you have not specified a password, it means that you have an incorrect password listed in some option file. Try the --no-defaults option as described in the previous item.

For information on changing passwords, see Section 4.14, “Assigning Account Passwords”.

If you have lost or forgotten the root password, see How to Reset the Root Password.

• localhost is a synonym for your local host name, and is also the default host to which clients try to connect if you specify no host explicitly.

You can use a --host=127.0.0.1 option to name the server host explicitly. This will make a TCP/IP connection to the local mysqld server. You can also use TCP/IP by specifying a --host option that uses the actual host name of the local host. In this case, the host name must be specified in a user table row on the server host, even though you are running the client program on the same host as the server.

• The Access denied error message tells you who you are trying to log in as, the client host from which you are trying to connect, and whether you were using a password. Normally, you should have one row in the user table that exactly matches the host name and user name that were given in the error message. For example, if you get an error message that contains using password: NO, it means that you tried to log in without a password.

• If you get an Access denied error when trying to connect to the database with mysql -u user_name, you may have a problem with the user table. Check this by executing mysql -u root mysql and issuing this SQL statement:

```
SELECT * FROM user;
```

The result should include a row with the Host and User columns matching your client’s host name and your MySQL user name.

• If the following error occurs when you try to connect from a host other than the one on which the MySQL server is running, it means that there is no row in the user table with a Host value that matches the client host:

```
Access denied for user 'root'@'some_hostname' (using password: YES)
```

You can fix this by setting up an account for the combination of client host name and user name that you are using when trying to connect.

If you do not know the IP address or host name of the machine from which you are connecting, you should put a row with '%' as the Host column value in the user table. After trying to connect from the client machine, use a SELECT USER() query to see how you really did connect. Then change the '%' in the user table row to the actual host name that shows up in the log. Otherwise, your system is left insecure because it permits connections from any host for the given user name.

On Linux, another reason that this error might occur is that you are using a binary MySQL version that is compiled with a different version of the glibc library than the one you are using. In this case, you should either upgrade your operating system or glibc, or download a source distribution of MySQL version and compile it yourself. A source RPM is normally trivial to compile and install, so this is not a big problem.

• If you specify a host name when trying to connect, but get an error message where the host name is not shown or is an IP address, it means that the MySQL server got an error when trying to resolve the IP address of the client host to a name:

shell> mysqladmin -u root -pxxxx -h some_hostname ver
Access denied for user 'root'@'some_hostname' (using password: YES)
If you try to connect as `root` and get the following error, it means that you do not have a row in the `user` table with a `User` column value of `root` and that `mysqld` cannot resolve the host name for your client:

```
Access denied for user ''@'unknown'
```

These errors indicate a DNS problem. To fix it, execute `mysqladmin flush-hosts` to reset the internal DNS host cache. See DNS Lookup Optimization and the Host Cache.

Some permanent solutions are:

- Determine what is wrong with your DNS server and fix it.
- Specify IP addresses rather than host names in the MySQL grant tables.
- Put an entry for the client machine name in `/etc/hosts` on Unix or `\windows\hosts` on Windows.
- Start `mysqld` with the `skip_name_resolve` system variable enabled.
- Start `mysqld` with the `--skip-host-cache` option.
- On Unix, if you are running the server and the client on the same machine, connect to `localhost`. For connections to `localhost`, MySQL programs attempt to connect to the local server by using a Unix socket file, unless there are connection parameters specified to ensure that the client makes a TCP/IP connection. For more information, see Connecting to the MySQL Server Using Command Options.
- On Windows, if you are running the server and the client on the same machine and the server supports named pipe connections, connect to the host name. (period). Connections to . use a named pipe rather than TCP/IP.
- If `mysql -u root` works but `mysql -h your_hostname -u root` results in `Access denied` (where `your_hostname` is the actual host name of the local host), you may not have the correct name for your host in the `user` table. A common problem here is that the `Host` value in the `user` table row specifies an unqualified host name, but your system's name resolution routines return a fully qualified domain name (or vice versa). For example, if you have a row with host 'pluto' in the `user` table, but your DNS tells MySQL that your host name is 'pluto.example.com', the row does not work. Try adding a row to the `user` table that contains the IP address of your host as the `Host` column value. (Alternatively, you could add a row to the `user` table with a `Host` value that contains a wildcard (for example, 'pluto.%'). However, use of `Host` values ending with % is insecure and is not recommended!)
- If `mysql -u user_name` works but `mysql -u user_name some_db` does not, you have not granted access to the given user for the database named `some_db`.
- If `mysql -u user_name` works when executed on the server host, but `mysql -h host_name -u user_name` does not work when executed on a remote client host, you have not enabled access to the server for the given user name from the remote host.
- If you cannot figure out why you get `Access denied`, remove from the `user` table all rows that have `Host` values containing wildcards (rows that contain '%' or '_' characters). A very common error is to insert a new row with `Host='%'` and `User='some_user'`, thinking that this enables you to specify `localhost` to connect from the same machine. The reason that this does not work is that the default privileges include a row with `Host='localhost'` and `User=''`. Because that row has a `Host` value 'localhost' that is more specific than '%', it is used in preference to the new row when connecting from `localhost`! The correct procedure is to insert a second row with `Host='localhost'` and `User='some_user'`, or to delete the row with `Host='localhost'` and `User=''`. After deleting the row, remember to issue a `FLUSH PRIVILEGES` statement to reload the grant tables. See also Section 4.6, “Access Control, Stage 1: Connection Verification”.

115
• If you are able to connect to the MySQL server, but get an Access denied message whenever you issue a `SELECT ... INTO OUTFILE` or `LOAD DATA` statement, your row in the `user` table does not have the `FILE` privilege enabled.

• If you change the grant tables directly (for example, by using `INSERT`, `UPDATE`, or `DELETE` statements) and your changes seem to be ignored, remember that you must execute a `FLUSH PRIVILEGES` statement or a `mysqladmin flush-privileges` command to cause the server to reload the privilege tables. Otherwise, your changes have no effect until the next time the server is restarted. Remember that after you change the root password with an `UPDATE` statement, you will not need to specify the new password until after you flush the privileges, because the server will not know you've changed the password yet!

• If your privileges seem to have changed in the middle of a session, it may be that a MySQL administrator has changed them. Reloading the grant tables affects new client connections, but it also affects existing connections as indicated in Section 4.13, "When Privilege Changes Take Effect".

• If you have access problems with a Perl, PHP, Python, or ODBC program, try to connect to the server with `mysql -u user_name db_name` or `mysql -u user_name -p password db_name`. If you are able to connect using the `mysql` client, the problem lies with your program, not with the access privileges. (There is no space between `-p` and the password; you can also use the `--password=` syntax to specify the password. If you use the `-p` or `--password` option with no password value, MySQL prompts for the password.)

• For testing purposes, start the `mysqld` server with the `--skip-grant-tables` option. Then you can change the MySQL grant tables and use the `SHOW GRANTS` statement to check whether your modifications have the desired effect. When you are satisfied with your changes, execute `mysqladmin flush-privileges` to tell the `mysqld` server to reload the privileges. This enables you to begin using the new grant table contents without stopping and restarting the server.

• If everything else fails, start the `mysqld` server with a debugging option (for example, `--debug=d,general,query`). This prints host and user information about attempted connections, as well as information about each command issued. See The DEBUG Package.

• If you have any other problems with the MySQL grant tables and ask on the MySQL Community Slack, always provide a dump of the MySQL grant tables. You can dump the tables with the `mysqldump mysql` command. To file a bug report, see the instructions at How to Report Bugs or Problems. In some cases, you may need to restart `mysqld` with `--skip-grant-tables` to run `mysqldump`.

4.22 SQL-Based Account Activity Auditing

Applications can use the following guidelines to perform SQL-based auditing that ties database activity to MySQL accounts.

MySQL accounts correspond to rows in the `mysql.user` system table. When a client connects successfully, the server authenticates the client to a particular row in this table. The `User` and `Host` column values in this row uniquely identify the account and correspond to the `'user_name'@'host_name'` format in which account names are written in SQL statements.

The account used to authenticate a client determines which privileges the client has. Normally, the `CURRENT_USER()` function can be invoked to determine which account this is for the client user. Its value is constructed from the `User` and `Host` columns of the `user` table row for the account.

However, there are circumstances under which the `CURRENT_USER()` value corresponds not to the client user but to a different account. This occurs in contexts when privilege checking is not based the client's account:

• Stored routines (procedures and functions) defined with the `SQL SECURITY DEFINER` characteristic

• Views defined with the `SQL SECURITY DEFINER` characteristic
• Triggers and events

In those contexts, privilege checking is done against the `DEFINER` account and `CURRENT_USER()` refers to that account, not to the account for the client who invoked the stored routine or view or who caused the trigger to activate. To determine the invoking user, you can call the `USER()` function, which returns a value indicating the actual user name provided by the client and the host from which the client connected. However, this value does not necessarily correspond directly to an account in the `user` table, because the `USER()` value never contains wildcards, whereas account values (as returned by `CURRENT_USER()`) may contain user name and host name wildcards.

For example, a blank user name matches any user, so an account of `''@'localhost'` enables clients to connect as an anonymous user from the local host with any user name. In this case, if a client connects as `user1` from the local host, `USER()` and `CURRENT_USER()` return different values:

```sql
mysql> SELECT USER(), CURRENT_USER();
+-----------------+----------------+
| USER()          | CURRENT_USER() |
+-----------------+----------------+
| user1@localhost | @localhost     |
+-----------------+----------------+
```

The host name part of an account can contain wildcards, too. If the host name contains a `'%'` or `'_'` pattern character or uses netmask notation, the account can be used for clients connecting from multiple hosts and the `CURRENT_USER()` value will not indicate which one. For example, the account `''user2''@'%.example.com'` can be used by `user2` to connect from any host in the `example.com` domain. If `user2` connects from `remote.example.com`, `USER()` and `CURRENT_USER()` return different values:

```sql
mysql> SELECT USER(), CURRENT_USER();
+--------------------------+---------------------+
| USER()                   | CURRENT_USER()      |
+--------------------------+---------------------+
| user2@remote.example.com | user2@%.example.com |
+--------------------------+---------------------+
```

If an application must invoke `USER()` for user auditing (for example, if it does auditing from within triggers) but must also be able to associate the `USER()` value with an account in the `user` table, it is necessary to avoid accounts that contain wildcards in the `User` or `Host` column. Specifically, do not permit `User` to be empty (which creates an anonymous-user account), and do not permit pattern characters or netmask notation in `Host` values. All accounts must have a nonempty `User` value and literal `Host` value.

With respect to the previous examples, the `''@'localhost'` and `''user2''@'%.example.com'` accounts should be changed not to use wildcards:

```sql
RENAME USER ''@'localhost' TO 'user1'@'localhost';
RENAME USER 'user2'@'%.example.com' TO 'user2'@'remote.example.com';
```

If `user2` must be able to connect from several hosts in the `example.com` domain, there should be a separate account for each host.

To extract the user name or host name part from a `CURRENT_USER()` or `USER()` value, use the `SUBSTRING_INDEX()` function:

```sql
mysql> SELECT SUBSTRING_INDEX(CURRENT_USER(), '@', 1);
+----------------+
| SUBSTRING_INDEX(CURRENT_USER(), '@', 1) |
+----------------+
| user1          |

mysql> SELECT SUBSTRING_INDEX(CURRENT_USER(), '@', -1);
+----------------+
| SUBSTRING_INDEX(CURRENT_USER(), '@', -1) |
+----------------+
| localhost      |
```
Chapter 5 Using Encrypted Connections

Table of Contents

5.1 Configuring MySQL to Use Encrypted Connections ................................................................. 120
5.2 Encrypted Connection TLS Protocols and Ciphers ................................................................. 126
5.3 Creating SSL and RSA Certificates and Keys ................................................................. 132
  5.3.1 Creating SSL and RSA Certificates and Keys using MySQL ........................................... 132
  5.3.2 Creating SSL Certificates and Keys Using openssl ..................................................... 135
  5.3.3 Creating RSA Keys Using openssl ................................................................................. 140
5.4 Connecting to MySQL Remotely from Windows with SSH .................................................. 140

With an unencrypted connection between the MySQL client and the server, someone with access to
the network could watch all your traffic and inspect the data being sent or received between client and
server.

When you must move information over a network in a secure fashion, an unencrypted connection
is unacceptable. To make any kind of data unreadable, use encryption. Encryption algorithms must
include security elements to resist many kinds of known attacks such as changing the order of
encrypted messages or replaying data twice.

MySQL supports encrypted connections between clients and the server using the TLS (Transport
Layer Security) protocol. TLS is sometimes referred to as SSL (Secure Sockets Layer) but MySQL
does not actually use the SSL protocol for encrypted connections because its encryption is weak (see
Section 5.2, “Encrypted Connection TLS Protocols and Ciphers”).

TLS uses encryption algorithms to ensure that data received over a public network can be trusted. It
has mechanisms to detect data change, loss, or replay. TLS also incorporates algorithms that provide
identity verification using the X.509 standard.

X.509 makes it possible to identify someone on the Internet. In basic terms, there should be some
entity called a “Certificate Authority” (or CA) that assigns electronic certificates to anyone who needs
them. Certificates rely on asymmetric encryption algorithms that have two encryption keys (a public key
and a secret key). A certificate owner can present the certificate to another party as proof of identity. A
certificate consists of its owner’s public key. Any data encrypted using this public key can be decrypted
only using the corresponding secret key, which is held by the owner of the certificate.

Support for encrypted connections in MySQL is provided using OpenSSL. For information about the
encryption protocols and ciphers that OpenSSL supports, see Section 5.2, “Encrypted Connection TLS
Protocols and Ciphers”.

Note

From MySQL 8.0.11 to 8.0.17, it was possible to compile MySQL using wolfSSL
as an alternative to OpenSSL. As of MySQL 8.0.18, support for wolfSSL is
removed and all MySQL builds use OpenSSL.

By default, MySQL programs attempt to connect using encryption if the server supports encrypted
connections, falling back to an unencrypted connection if an encrypted connection cannot be
established. For information about options that affect use of encrypted connections, see Section 5.1,
“Configuring MySQL to Use Encrypted Connections” and Command Options for Encrypted
Connections.

MySQL performs encryption on a per-connection basis, and use of encryption for a given user can be
optional or mandatory. This enables you to choose an encrypted or unencrypted connection according
to the requirements of individual applications. For information on how to require users to use encrypted
connections, see the discussion of the REQUIRE clause of the CREATE USER statement in CREATE
USER Statement. See also the description of the require_secure_transport system variable at
Server System Variables.
Encrypted connections can be used between master and slave replication servers. See Setting Up Replication to Use Encrypted Connections.

For information about using encrypted connections from the MySQL C API, see C API Encrypted Connection Support.

It is also possible to connect using encryption from within an SSH connection to the MySQL server host. For an example, see Section 5.4, “Connecting to MySQL Remotely from Windows with SSH”.

5.1 Configuring MySQL to Use Encrypted Connections

Several configuration parameters are available to indicate whether to use encrypted connections, and to specify the appropriate certificate and key files. This section provides general guidance about configuring the server and clients for encrypted connections:

- Server-Side Startup Configuration for Encrypted Connections
- Server-Side Runtime Configuration and Monitoring for Encrypted Connections
- Client-Side Configuration for Encrypted Connections
- Configuring Encrypted Connections as Mandatory

Encrypted connections also can be used in these contexts:

- Between master and slave replication servers. See Setting Up Replication to Use Encrypted Connections.
- Among Group Replication servers. See Group Replication Secure Socket Layer (SSL) Support.
- By client programs that are based on the MySQL C API. See C API Encrypted Connection Support.

Instructions for creating any required certificate and key files are available in Section 5.3, “Creating SSL and RSA Certificates and Keys”.

Server-Side Startup Configuration for Encrypted Connections

On the server side, the --ssl option specifies that the server permits but does not require encrypted connections. This option is enabled by default, so it need not be specified explicitly.

To require that clients connect using encrypted connections, enable the require_secure_transport system variable. See Configuring Encrypted Connections as Mandatory.

These system variables on the server side specify the certificate and key files the server uses when permitting clients to establish encrypted connections:

- **ssl_ca**: The path name of the Certificate Authority (CA) certificate file. *(ssl_capath is similar but specifies the path name of a directory of CA certificate files.)*

- **ssl_cert**: The path name of the server public key certificate file. This certificate can be sent to the client and authenticated against the CA certificate that it has.

- **ssl_key**: The path name of the server private key file.

For example, to enable the server for encrypted connections, start it with these lines in the my.cnf file, changing the file names as necessary:

```ini
[mysqld]
ssl_ca=ca.pem
ssl_cert=server-cert.pem
ssl_key=server-key.pem
```

To specify in addition that clients are required to use encrypted connections, enable the require_secure_transport system variable:
Each certificate and key system variable names a file in PEM format. Should you need to create
the required certificate and key files, see Section 5.3, “Creating SSL and RSA Certificates and
Keys”. MySQL servers compiled using OpenSSL can generate missing certificate and key files
automatically at startup. See Section 5.3.1, “Creating SSL and RSA Certificates and Keys using
MySQL”. Alternatively, if you have a MySQL source distribution, you can test your setup using the
demonstration certificate and key files in its mysql-test/std_data directory.

The server performs certificate and key file autodiscovery. If no explicit SSL options are given other
than --ssl (possibly along with ssl_cipher) to configure encrypted connections, the server
attempts to enable encrypted connection support automatically at startup:

- If the server discovers valid certificate and key files named ca.pem, server-cert.pem, and
  server-key.pem in the data directory, it enables support for encrypted connections by clients. (The
  files need not have been generated automatically; what matters is that they have those names and
  are valid.)
- If the server does not find valid certificate and key files in the data directory, it continues executing
  but without support for encrypted connections.

If the server automatically enables encrypted connection support, it writes a note to the error
log. If the server discovers that the CA certificate is self-signed, it writes a warning to the error
log. (The certificate is self-signed if created automatically by the server or manually using
mysql_ssl_rsa_setup.)

MySQL also provides these system variables for server-side SSL control:

- **ssl_cipher**: The list of permissible ciphers for connection encryption.
- **ssl_crl**: The path name of the file containing certificate revocation lists. (ssl_crlpath is similar
  but specifies the path name of a directory of certificate revocation-list files.)
- **tls_version,tls_ciphersuites**: Which encryption protocols and ciphersuites the server
  permits for encrypted connections; see Section 5.2, “Encrypted Connection TLS Protocols and
  Ciphers”. For example, you can set tls_version to prevent clients from using less-secure
  protocols.

**Server-Side Runtime Configuration and Monitoring for Encrypted Connections**

As of MySQL 8.0.16, the ssl_xxx and tls_xxx encryption-related system variables can be set at
runtime. If changed with SET GLOBAL, the new values apply to connections established subsequent
to the change, and apply only until server restart. If changed with SET PERSIST, the new values also
apply to subsequent server restarts. See SET Syntax for Variable Assignment.

In addition, as of MySQL 8.0.16, the TLS context the server uses for new connections is reconfigurable
at runtime. This capability may be useful, for example, to avoid restarting a MySQL server that has
been running so long that its SSL certificate has expired.

The server creates the initial TLS context from the values that the context-related system variables
have at startup. It also initializes a set of context-related status variables to indicate the values used
in the context. The following table shows the system variables that define the TLS context and the
corresponding status variables that indicate the currently active context values.

<table>
<thead>
<tr>
<th>System Variable Name</th>
<th>Status Variable Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssl_ca</td>
<td>Current_tls_ca</td>
</tr>
</tbody>
</table>
To reconfigure the TLS context at runtime, use this procedure:

1. For any TLS context-related system variables that should be changed, set them to their new values.

2. Execute `ALTER INSTANCE RELOAD TLS`. This statement reconfigures the active TLS context from the current values of the TLS context-related system variables. It also sets the context-related status variables to reflect the new active context values. The statement requires the `CONNECTION_ADMIN` privilege.

3. New connections established after execution of `ALTER INSTANCE RELOAD TLS` use the new TLS context. Existing connections remain unaffected. If existing connections should be terminated, use the `KILL` statement.

The members of each pair of system and status variables may have different values temporarily due to the way the reconfiguration procedure works:

- Changes to the system variables prior to `ALTER INSTANCE RELOAD TLS` do not change the TLS context. At this point, those changes have no effect on new connections, and corresponding context-related system and status variables may have different values. This enables you to make any changes required to the system variables, then update the active TLS context atomically with `ALTER INSTANCE RELOAD TLS` after all system variable changes have been made.

- After `ALTER INSTANCE RELOAD TLS`, corresponding system and status variables have the same values. This remains true until the next change to the system variables.

In some cases, `ALTER INSTANCE RELOAD TLS` by itself may suffice to reconfigure the TLS context, without changing any system variables. Suppose that the certificate in the file named by `ssl_cert` has expired. It is sufficient to replace the existing file contents with a nonexpired certificate and execute `ALTER INSTANCE RELOAD TLS` to cause the new file contents to be read and used for new connections.

By default, the `RELOAD TLS` action rolls back with an error and has no effect if the configuration values do not permit creation of the new TLS context. The previous context values continue to be used for new connections. If the optional `NO ROLLBACK ON ERROR` clause is given and the new context cannot be created, rollback does not occur. Instead, a warning is generated and encryption is disabled for new connections on the interface to which the statement applies.

The server-side `--ssl` option has an effect only at server startup on whether the server accepts encrypted connections. It is ignored by, and has no effect on the operation of, `ALTER INSTANCE RELOAD TLS`. For example, you can use `--ssl=OFF` to start the server with encrypted connections disabled, then reconfigure TLS and execute `ALTER INSTANCE RELOAD TLS` to enable encrypted connections at runtime.

`ALTER INSTANCE RELOAD TLS` changes only the TLS context the server itself uses for new connections. It does not affect the TLS context used by other enabled server plugins or components such as X Plugin or Group Replication. If you want to apply the reconfiguration to Group Replication's group communication connections, which take their settings from the server's TLS context,
related system variables, you must issue `STOP GROUP_REPLICATION` followed by `START GROUP_REPLICATION` to stop and restart Group Replication.

**Note**

Prior to MySQL 8.0.16, the TLS context-related system variables are not dynamic. They can be set at server startup, but cannot be changed thereafter. These system variables therefore determine the TLS context values the server uses for all new connections.

### Client-Side Configuration for Encrypted Connections

For a complete list of client options related to establishment of encrypted connections, see [Command Options for Encrypted Connections](#).

By default, MySQL client programs attempt to establish an encrypted connection if the server supports encrypted connections, with further control available through the `--ssl-mode` option:

- In the absence of an `--ssl-mode` option, clients attempt to connect using encryption, falling back to an unencrypted connection if an encrypted connection cannot be established. This is also the behavior with an explicit `--ssl-mode=PREFERRED` option.
- With `--ssl-mode=REQUIRED`, clients require an encrypted connection and fail if one cannot be established.
- With `--ssl-mode=DISABLED`, clients use an unencrypted connection.
- With `--ssl-mode=VERIFY_CA` or `--ssl-mode=VERIFY_IDENTITY`, clients require an encrypted connection, and also perform verification against the server CA certificate and (with `VERIFY_IDENTITY`) against the server host name in its certificate.

Attempts to establish an unencrypted connection fail if the `require_secure_transport` system variable is enabled on the server side to cause the server to require encrypted connections. See [Configuring Encrypted Connections as Mandatory](#).

The following options on the client side identify the certificate and key files clients use when establishing encrypted connections to the server. They are similar to the `ssl_ca`, `ssl_cert`, and `ssl_key` system variables used on the server side, but `--ssl-cert` and `--ssl-key` identify the client public and private key:

- `--ssl-ca`: The path name of the Certificate Authority (CA) certificate file. This option, if used, must specify the same certificate used by the server. (`--ssl-capath` is similar but specifies the path name of a directory of CA certificate files.)
- `--ssl-cert`: The path name of the client public key certificate file.
- `--ssl-key`: The path name of the client private key file.

For additional security relative to that provided by the default encryption, clients can supply a CA certificate matching the one used by the server and enable host name identity verification. In this way, the server and client place their trust in the same CA certificate and the client verifies that the host to which it connected is the one intended:

- To specify the CA certificate, use `--ssl-ca` (or `--ssl-capath`), and specify `--ssl-mode=VERIFY_CA`.
- To enable host name identity verification as well, use `--ssl-mode=VERIFY_IDENTITY` rather than `--ssl-mode=VERIFY_CA`.

**Note**

Host name identity verification with `VERIFY_IDENTITY` does not work with self-signed certificates that are created automatically by the server or manually
Client-Side Configuration for Encrypted Connections

using `mysql_ssl_rsa_setup` (see Section 5.3.1, “Creating SSL and RSA Certificates and Keys using MySQL”). Such self-signed certificates do not contain the server name as the Common Name value.

Host name identity verification also does not work with certificates that specify the Common Name using wildcards because that name is compared verbatim to the server name.

MySQL also provides these options for client-side SSL control:

- **`--ssl-cipher`**: The list of permissible ciphers for connection encryption.
- **`--ssl-crl`**: The path name of the file containing certificate revocation lists. (`--ssl-crlpath` is similar but specifies the path name of a directory of certificate revocation-list files.)
- **`--tls-version, --tls-ciphersuites`**: The permitted encryption protocols and ciphersuites; see Section 5.2, "Encrypted Connection TLS Protocols and Ciphers".

Depending on the encryption requirements of the MySQL account used by a client, the client may be required to specify certain options to connect using encryption to the MySQL server.

Suppose that you want to connect using an account that has no special encryption requirements or that was created using a `CREATE USER` statement that included the `REQUIRE SSL` clause. Assuming that the server supports encrypted connections, a client can connect using encryption with no `--ssl-mode` option or with an explicit `--ssl-mode=PREFERRED` option:

```
mysql
```

Or:

```
mysql --ssl-mode=PREFERRED
```

For an account created with a `REQUIRE SSL` clause, the connection attempt fails if an encrypted connection cannot be established. For an account with no special encryption requirements, the attempt falls back to an unencrypted connection if an encrypted connection cannot be established. To prevent fallback and fail if an encrypted connection cannot be obtained, connect like this:

```
mysql --ssl-mode=REQUIRED
```

If the account has more stringent security requirements, other options must be specified to establish an encrypted connection:

- For accounts created with a `REQUIRE X509` clause, clients must specify at least `--ssl-cert` and `--ssl-key`. In addition, `--ssl-ca (or --ssl-capath)` is recommended so that the public certificate provided by the server can be verified. For example (enter the command on a single line):

```
mysql --ssl-ca=ca.pem
--ssl-cert=client-cert.pem
--ssl-key=client-key.pem
```

- For accounts created with a `REQUIRE ISSUER` or `REQUIRE SUBJECT` clause, the encryption requirements are the same as for `REQUIRE X509`, but the certificate must match the issue or subject, respectively, specified in the account definition.

For additional information about the `REQUIRE` clause, see CREATE USER Statement.

To prevent use of encryption and override other `--ssl-xxx` options, invoke the client program with `--ssl-mode=DISABLED`:

```
mysql --ssl-mode=DISABLED
```

To determine whether the current connection with the server uses encryption, check the session value of the `Ssl_cipher` status variable. If the value is empty, the connection is not encrypted. Otherwise, the connection is encrypted and the value indicates the encryption cipher. For example:

```
mysql> SHOW SESSION STATUS LIKE 'Ssl_cipher';
```
Configuring Encrypted Connections as Mandatory

For some MySQL deployments it may be not only desirable but mandatory to use encrypted connections (for example, to satisfy regulatory requirements). This section discusses configuration settings that enable you to do this. These levels of control are available:

- You can configure the server to require that clients connect using encrypted connections.

- You can invoke individual client programs to require an encrypted connection, even if the server permits but does not require encryption.

- You can configure individual MySQL accounts to be usable only over encrypted connections.

To require that clients connect using encrypted connections, enable the `require_secure_transport` system variable. For example, put these lines in the server `my.cnf` file:

```
[mysqld]
require_secure_transport=ON
```

Alternatively, to set and persist the value at runtime, use this statement:

```
SET PERSIST require_secure_transport=ON;
```

**SET PERSIST** sets the value for the running MySQL instance. It also saves the value, causing it to be used for subsequent server restarts. See **SET Syntax for Variable Assignment**.

With `require_secure_transport` enabled, client connections to the server are required to use some form of secure transport, and the server permits only TCP/IP connections that use SSL, or connections that use a socket file (on Unix) or shared memory (on Windows). The server rejects nonsecure connection attempts, which fail with an `ER_SECURE_TRANSPORT_REQUIRED` error.

To invoke a client program such that it requires an encrypted connection whether or not the server requires encryption, use an `--ssl-mode` option value of `REQUIRED`, `VERIFY_CA`, or `VERIFY_IDENTITY`. For example:

```
mysql --ssl-mode=REQUIRED
mysqldump --ssl-mode=VERIFY_CA
mysqladmin --ssl-mode=VERIFY_IDENTITY
```

To configure a MySQL account to be usable only over encrypted connections, include a `REQUIRE` clause in the `CREATE USER` statement that creates the account, specifying in that clause the encryption characteristics you require. For example, to require an encrypted connection and the use of a valid X.509 certificate, use `REQUIRE X509`:

```
CREATE USER 'jeffrey'@'localhost' REQUIRE X509;
```

For additional information about the `REQUIRE` clause, see **CREATE USER Statement**.
To modify existing accounts that have no encryption requirements, use the `ALTER USER` statement.

## 5.2 Encrypted Connection TLS Protocols and Ciphers

MySQL supports multiple TLS protocols and ciphers, and enables configuring which protocols and ciphers to permit for encrypted connections. It is also possible to determine which protocol and cipher the current session uses.

**Supported Connection TLS Protocols**

**Connection TLS Protocol Configuration**

**Connection Cipher Configuration**

**Connection TLS Protocol Negotiation**

**Monitoring Current Client Session TLS Protocol and Cipher**

### Supported Connection TLS Protocols

MySQL supports encrypted connections using the TLSv1, TLSv1.1, TLSv1.2, and TLSv1.3 protocols, listed in order from less secure to more secure. The set of protocols actually permitted for connections is subject to multiple factors:

- **MySQL configuration.** Permitted TLS protocols can be configured on both the server side and client side to include only a subset of the supported TLS protocols. The configuration on both sides must include at least one protocol in common or connection attempts cannot negotiate a protocol to use. For details, see **Connection TLS Protocol Negotiation**.

- **System-wide host configuration.** The host system may permit only certain TLS protocols, which means that MySQL connections cannot use nonpermitted protocols even if MySQL itself permits them:
  
  - Suppose that MySQL configuration permits TLSv1, TLSv1.1, and TLSv1.2, but your host system configuration permits only connections that use TLSv1.2 or higher. In this case, you cannot establish MySQL connections that use TLSv1 or TLSv1.1, even though MySQL is configured to permit them, because the host system does not permit them.
  
  - If MySQL configuration permits TLSv1, TLSv1.1, and TLSv1.2, but your host system configuration permits only connections that use TLSv1.3 or higher, you cannot establish MySQL connections at all, because no protocol permitted by MySQL is permitted by the host system.

Workarounds for this issue include:

- **Change the system-wide host configuration to permit additional TLS protocols.** Consult your operating system documentation for instructions. For example, your system may have an `/etc/ssl/openssl.cnf` file that contains these lines to restrict TLS protocols to TLSv1.2 or higher:

```
[system_default_sect]
MinProtocol = TLSv1.2
```

Changing the value to a lower protocol version or `None` makes the system more permissive. This workaround has the disadvantage that permitting lower (less secure) protocols may have adverse security consequences.

- **If you cannot or prefer not to change the host system TLS configuration,** change MySQL applications to use higher (more secure) TLS protocols that are permitted by the host system. This may not be possible for older versions of MySQL that support only lower protocol versions. For example, TLSv1 is the only supported protocol prior to MySQL 5.6.46, so attempts to connect to a pre-5.6.46 server fail even if the client is from a newer MySQL version that supports higher protocol versions. In such cases, an upgrade to a version of MySQL that supports additional TLS versions may be required.
Connection TLS Protocol Configuration

- The SSL library. If the SSL library does not support a particular protocol, neither does MySQL, and any parts of the following discussion that specify that protocol do not apply.

**Note**
Support for the TLSv1.3 protocol is available as of MySQL 8.0.16 (as of MySQL 8.0.18 for the Group Replication component). In addition, to use TLSv1.3, both the MySQL server and the client application must be compiled using OpenSSL 1.1.1 or higher.

Connection TLS Protocol Configuration

On the server side, the value of the `tls_version` system variable determines which TLS protocols a MySQL server permits for encrypted connections. The `tls_version` value applies to connections from clients, regular master/slave replication connections where this server instance is the master, Group Replication group communication connections, and Group Replication distributed recovery connections where this server instance is the donor. The variable value is a list of one or more comma-separated protocol versions from this list (not case-sensitive): TLSv1, TLSv1.1, TLSv1.2, and (if available) TLSV1.3. By default, this variable lists all protocols supported by the SSL library used to compile MySQL. To determine the value of `tls_version` at runtime, use this statement:

```sql
mysql> SHOW GLOBAL VARIABLES LIKE 'tls_version';
+---------------+-----------------------+
| Variable_name | Value                 |
|---------------+-----------------------+
| tls_version   | TLSv1,TLSv1.1,TLSv1.2 |
+---------------+-----------------------+
```

To change the value of `tls_version`, set it at server startup. For example, to permit connections that use the TLSv1.1 or TLSv1.2 protocol, but prohibit connections that use the less-secure TLSv1 protocol, use these lines in the server `my.cnf` file:

```
[mysqld]
tls_version=TLSv1.1,TLSv1.2
```

To be even more restrictive and permit only TLSv1.2 connections, set `tls_version` like this:

```
[mysqld]
tls_version=TLSv1.2
```

As of MySQL 8.0.16, `tls_version` can also be changed at runtime. See Server-Side Runtime Configuration and Monitoring for Encrypted Connections.

On the client side, the `--tls-version` option specifies which TLS protocols a client program permits for connections to the server. The format of the option value is the same as for the `tls_version` system variable described previously (a list of one or more comma-separated protocol versions).

For master/slave replication connections where this server instance is the slave, the `MASTER_TLS_VERSION` option for the `CHANGE MASTER TO` statement specifies which TLS protocols the replication slave permits for connections to the master. The format of the option value is the same as for the `tls_version` system variable described previously. See Setting Up Replication to Use Encrypted Connections.

The protocols that can be specified for `MASTER_TLS_VERSION` depend on the SSL library. This option is independent of and not affected by the server `tls_version` value. For example, a server that acts as a replication slave can be configured with `tls_version` set to TLSv1.3 to permit only incoming connections that use TLSv1.3, but also configured with `MASTER_TLS_VERSION` set to TLSv1.2 to permit only TLSv1.2 for outgoing slave connections to the master.

For Group Replication distributed recovery connections where this server instance is the joining member that initiates distributed recovery (that is, the client), the `group_replication_recovery_tls_version` system variable specifies which protocols are permitted by the client. This option is independent of and not affected by the server `tls_version` value.
value, which applies when this server instance is the donor. A Group Replication server generally participates in distributed recovery both as a donor and as a joining member over the course of its group membership, so both these system variables should be set. See Group Replication Secure Socket Layer (SSL) Support.

TLS protocol configuration affects which protocol a given connection uses, as described in Connection TLS Protocol Negotiation.

Permitted protocols should be chosen such as not to leave “holes” in the list. For example, these server configuration values do not have holes:

- `tls_version=TLSv1,TLSv1.1,TLSv1.2,TLSv1.3`
- `tls_version=TLSv1.2,TLSv1.3`
- `tls_version=TLSv1.3`

These values do have holes and should not be used:

- `tls_version=TLSv1,TLSv1.2` (TLSv1.1 is missing)
- `tls_version=TLSv1.1,TLSv1.3` (TLSv1.2 is missing)

The prohibition on holes also applies in other configuration contexts, such as for clients or replication slaves.

The list of permitted protocols should not be empty. If you set a TLS version parameter to the empty string, encrypted connections cannot be established:

- `tls_version`: The server does not permit encrypted incoming connections.
- `--tls-version`: The client does not permit encrypted outgoing connections to the server.
- `MASTER_TLS_VERSION`: The replication slave does not permit encrypted outgoing connections to the master.

### Connection Cipher Configuration

A default set of ciphers applies to encrypted connections, which can be overridden by explicitly configuring the permitted ciphers. During connection establishment, both sides of a connection must permit some cipher in common or the connection fails. Of the permitted ciphers common to both sides, the SSL library chooses the one supported by the provided certificate that has the highest priority.

To specify a cipher or ciphers applicable for encrypted connections that use TLS protocols up through TLSv1.2:

- Set the `ssl_cipher` system variable on the server side, and use the `--ssl-cipher` option for client programs.

- For regular master/slave replication connections, where this server instance is the master, set the `ssl_cipher` system variable. Where this server instance is the slave, use the `MASTER_SSL_CIPHER` option for the `CHANGE MASTER TO` statement. See Setting Up Replication to Use Encrypted Connections.

- For a Group Replication group member, for Group Replication group communication connections and also for Group Replication distributed recovery connections where this server instance is the donor, set the `ssl_cipher` system variable. For Group Replication distributed recovery connections where this server instance is the joining member, use the `group_replication_recovery_ssl_cipher` system variable. See Group Replication Secure Socket Layer (SSL) Support.

For encrypted connections that use TLSv1.3, OpenSSL 1.1.1 and higher supports the following ciphersuites, the first three of which are enabled by default:

- `TLS_AES_128_GCM_SHA256`
- `TLS_AES_256_GCM_SHA384`
- `TLS_CHACHA20_POLY1305_SHA256`
To configure the permitted TLSv1.3 ciphersuites explicitly, set the following parameters. In each case, the configuration value is a list of zero or more colon-separated ciphersuite names.

- On the server side, use the `tls_ciphersuites` system variable. If this variable is not set, its default value is `NULL`, which means that the server permits the default set of ciphersuites. If the variable is set to the empty string, no ciphersuites are enabled and encrypted connections cannot be established.

- On the client side, use the `--tls-ciphersuites` option. If this option is not set, the client permits the default set of ciphersuites. If the option is set to the empty string, no ciphersuites are enabled and encrypted connections cannot be established.

- For regular master/slave replication connections, where this server instance is the master, use the `tls_ciphersuites` system variable. Where this server instance is the slave, use the `MASTER_TLS_CIPHERSUITES` option for the `CHANGE MASTER TO` statement. See Setting Up Replication to Use Encrypted Connections.

- For a Group Replication group member, for Group Replication group communication connections and also for Group Replication distributed recovery connections where this server instance is the donor, use the `tls_ciphersuites` system variable. For Group Replication distributed recovery connections where this server instance is the joining member, use the `group_replication_recovery_tls_ciphersuites` system variable. See Group Replication Secure Socket Layer (SSL) Support.

### Note

Ciphersuite support is available as of MySQL 8.0.16, but requires that both the MySQL server and the client application be compiled using OpenSSL 1.1.1 or higher.

In MySQL 8.0.16 through 8.0.18, the `MASTER_TLS_CIPHERSUITES` option for the `CHANGE MASTER TO` statement and the `group_replication_recovery_tls_ciphersuites` system variable are not available. In these releases, if TLSv1.3 is used for master/slave replication connections, or in Group Replication for distributed recovery (supported from MySQL 8.0.18), the replication master or Group Replication donor servers must permit the use of at least one TLSv1.3 ciphersuite that is enabled by default. From MySQL 8.0.19, you can use the options to configure client support for any selection of ciphersuites, including only non-default ciphersuites if you want.

A given cipher may work only with particular TLS protocols, which affects the TLS protocol negotiation process. See Connection TLS Protocol Negotiation.

To determine which ciphers a given server supports, check the session value of the `Ssl_cipher_list` status variable:

```
SHOW SESSION STATUS LIKE 'Ssl_cipher_list';
```

The `Ssl_cipher_list` status variable lists the possible SSL ciphers (empty for non-SSL connections). If MySQL supports TLSv1.3, the value includes the possible TLSv1.3 ciphersuites.

For encrypted connections that use TLSv1.3, MySQL uses the SSL library default ciphersuite list.

For encrypted connections that use TLS protocols up through TLSv1.2, MySQL passes the following default cipher list to the SSL library:

```
ECDHE-ECDSA-AES128-GCM-SHA256
ECDHE-ECDSA-AES256-GCM-SHA384
ECDHE-RSA-AES128-GCM-SHA256
ECDHE-RSA-AES256-GCM-SHA384
```
These cipher restrictions are in place:

- The following ciphers are permanently restricted:

  - !DHE-DSS-DES-CBC3-SHA
  - !DHE-RSA-DES-CBC3-SHA
Connection TLS Protocol Negotiation

Connection attempts in MySQL negotiate use of the highest TLS protocol version available on both sides for which a protocol-compatible encryption cipher is available on both sides. The negotiation process depends on factors such as the SSL library used to compile the server and client, the TLS protocol and encryption cipher configuration, and which key size is used:

- For a connection attempt to succeed, the server and client TLS protocol configuration must permit some protocol in common.
- Similarly, the server and client encryption cipher configuration must permit some cipher in common. A given cipher may work only with particular TLS protocols, so a protocol available to the negotiation process is not chosen unless there is also a compatible cipher.
- If TLSv1.3 is available, it is used if possible. (This means that server and client configuration both must permit TLSv1.3, and both must also permit some TLSv1.3-compatible encryption cipher.) Otherwise, MySQL continues through the list of available protocols, using TLSv1.2 if possible, and so forth. Negotiation proceeds from more secure protocols to less secure. Negotiation order is independent of the order in which protocols are configured. For example, negotiation order is the same regardless of whether `tls_version` has a value of `TLSv1,TLSv1.1,TLSv1.2,TLSv1.3` or `TLSv1.3,TLSv1.2,TLSv1.1,TLSv1`.

- TLSv1.2 does not work with all ciphers that have a key size of 512 bits or less. To use this protocol with such a key, set the `ssl_cipher` system variable on the server side or use the `--ssl-cipher` client option to specify the cipher name explicitly:

  - AES128-SHA
  - AES128-SHA256
  - AES256-SHA
  - AES256-SHA256
  - CAMELLIA128-SHA
  - CAMELLIA256-SHA
  - DES-CBC3-SHA
  - DHE-RSA-AES256-SHA
  - RC4-MD5
  - RC4-SHA
  - SEED-SHA

- For better security, use a certificate with an RSA key size of at least 2048 bits.

If the server and client do not have a permitted protocol in common, and a protocol-compatible cipher in common, the server terminates the connection request. Examples:

- If the server is configured with `tls_version=TLSv1.1,TLSv1.2`: 

If the server is started with the `ssl_cert` system variable set to a certificate that uses any of the preceding restricted ciphers or cipher categories, the server starts with support for encrypted connections disabled.
• Connection attempts fail for clients invoked with \texttt{--tls-version=TLSv1}, and for older clients that support only TLSv1.

• Similarly, connection attempts fail for replication slaves configured with \texttt{MASTER_TLS_VERSION = 'TLSv1'}, and for older slaves that support only TLSv1.

• If the server is configured with \texttt{tls_version=TLSv1} or is an older server that supports only TLSv1:
  • Connection attempts fail for clients invoked with \texttt{--tls-version=TLSv1.1,TLSv1.2}.
  • Similarly, connection attempts fail for replication slaves configured with \texttt{MASTER_TLS_VERSION = 'TLSv1.1,TLSv1.2'}.

MySQL permits specifying a list of protocols to support. This list is passed directly down to the underlying SSL library and is ultimately up to that library what protocols it actually enables from the supplied list. Please refer to the MySQL source code and the OpenSSL \texttt{SSL_CTX_new()} documentation for information about how the SSL library handles this.

\section*{Monitoring Current Client Session TLS Protocol and Cipher}

To determine which encryption TLS protocol and cipher the current client session uses, check the session values of the \texttt{Ssl_version} and \texttt{Ssl_cipher} status variables:

\begin{verbatim}
mysql> SELECT * FROM performance_schema.session_status
WHERE VARIABLE_NAME IN ('Ssl_version','Ssl_cipher');
+---------------+---------------------------+
| VARIABLE_NAME | VARIABLE_VALUE            |
+---------------+---------------------------+
| Ssl_cipher    | DHE-RSA-AES128-GCM-SHA256 |
| Ssl_version   | TLSv1.2                   |
+---------------+---------------------------+
\end{verbatim}

If the connection is not encrypted, both variables have an empty value.

\section*{5.3 Creating SSL and RSA Certificates and Keys}

The following discussion describes how to create the files required for SSL and RSA support in MySQL. File creation can be performed using facilities provided by MySQL itself, or by invoking the \texttt{openssl} command directly.

SSL certificate and key files enable MySQL to support encrypted connections using SSL. See Section 5.1, “Configuring MySQL to Use Encrypted Connections”.

RSA key files enable MySQL to support secure password exchange over unencrypted connections for accounts authenticated by the \texttt{sha256_password} or \texttt{caching_sha2_password} plugin. See Section 6.1.3, “SHA-256 Pluggable Authentication”, and Section 6.1.2, “Caching SHA-2 Pluggable Authentication”.

\subsection*{5.3.1 Creating SSL and RSA Certificates and Keys using MySQL}

MySQL provides these ways to create the SSL certificate and key files and RSA key-pair files required to support encrypted connections using SSL and secure password exchange using RSA over unencrypted connections, if those files are missing:

• The server can autogenerate these files at startup, for MySQL distributions compiled using OpenSSL.

• Users can invoke the \texttt{mysql_ssl_rsa_setup} utility manually.

• For some distribution types, such as RPM packages, \texttt{mysql_ssl_rsa_setup} invocation occurs during data directory initialization. In this case, the MySQL distribution need not have been compiled using OpenSSL as long as the \texttt{openssl} command is available.
Creating SSL and RSA Certificates and Keys using MySQL

Important
Server autogeneration and `mysql_ssl_rsa_setup` help lower the barrier to using SSL by making it easier to generate the required files. However, certificates generated by these methods are self-signed, which may not be very secure. After you gain experience using such files, consider obtaining certificate/key material from a registered certificate authority.

- Automatic SSL and RSA File Generation
- Manual SSL and RSA File Generation Using `mysql_ssl_rsa_setup`
- SSL and RSA File Characteristics

Automatic SSL and RSA File Generation

For MySQL distributions compiled using OpenSSL, the MySQL server has the capability of automatically generating missing SSL and RSA files at startup. The `auto_generate_certs`, `sha256_password_auto_generate_rsa_keys`, and `caching_sha2_password_auto_generate_rsa_keys` system variables control automatic generation of these files. These variables are enabled by default. They can be enabled at startup and inspected but not set at runtime.

At startup, the server automatically generates server-side and client-side SSL certificate and key files in the data directory if the `auto_generate_certs` system variable is enabled, no SSL options other than `--ssl` are specified, and the server-side SSL files are missing from the data directory. These files enable encrypted client connections using SSL; see Section 5.1, “Configuring MySQL to Use Encrypted Connections”.

1. The server checks the data directory for SSL files with the following names:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ca.pem</td>
<td>Self-signed CA certificate</td>
</tr>
<tr>
<td>server-cert.pem</td>
<td>Server certificate</td>
</tr>
<tr>
<td>server-key.pem</td>
<td>Server private key</td>
</tr>
</tbody>
</table>

2. If any of those files are present, the server creates no SSL files. Otherwise, it creates them, plus some additional files:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ca.pem</td>
<td>CA private key</td>
</tr>
<tr>
<td>ca-key.pem</td>
<td>CA private key</td>
</tr>
<tr>
<td>server-cert.pem</td>
<td>Server certificate</td>
</tr>
<tr>
<td>server-key.pem</td>
<td>Server private key</td>
</tr>
<tr>
<td>client-cert.pem</td>
<td>Client certificate</td>
</tr>
<tr>
<td>client-key.pem</td>
<td>Client private key</td>
</tr>
</tbody>
</table>

3. If the server autogenerates SSL files, it uses the names of the `ca.pem`, `server-cert.pem`, and `server-key.pem` files to set the corresponding system variables (`ssl_ca`, `ssl_cert`, `ssl_key`).

   At startup, the server automatically generates RSA private/public key-pair files in the data directory if all of these conditions are true: The `sha256_password_auto_generate_rsa_keys` or `caching_sha2_password_auto_generate_rsa_keys` system variable is enabled; no RSA options are specified; the RSA files are missing from the data directory. These key-pair files enable secure password exchange using RSA over unencrypted connections for accounts authenticated by the `sha256_password` or `caching_sha2_password` plugin; see Section 6.1.3, “SHA-256 Pluggable Authentication”, and Section 6.1.2, “Caching SHA-2 Pluggable Authentication”.

   1. The server checks the data directory for RSA files with the following names:

      | File       | Description                        |
      |------------|------------------------------------|
      | private_key.pem | Private member of private/public key pair |
      | public_key.pem  | Public member of private/public key pair |

   2. If any of these files are present, the server creates no RSA files. Otherwise, it creates them.

   3. If the server autogenerates the RSA files, it uses their names to set the corresponding system variables (`sha256_password_private_key_path` and \ldots)
Creating SSL and RSA Certificates and Keys using MySQL

- \texttt{sha256\_password\_public\_key\_path};\texttt{caching\_sha2\_password\_private\_key\_path} and \texttt{caching\_sha2\_password\_public\_key\_path}).

**Manual SSL and RSA File Generation Using mysql\_ssl\_rsa\_setup**

MySQL distributions include a \texttt{mysql\_ssl\_rsa\_setup} utility that can be invoked manually to generate SSL and RSA files. This utility is included with all MySQL distributions, but it does require that the \texttt{openssl} command be available. For usage instructions, see \texttt{mysql\_ssl\_rsa\_setup — Create SSL/RSA Files}.

**SSL and RSA File Characteristics**

SSL and RSA files created automatically by the server or by invoking \texttt{mysql\_ssl\_rsa\_setup} have these characteristics:

- SSL and RSA keys are have a size of 2048 bits.
- The SSL CA certificate is self signed.
- The SSL server and client certificates are signed with the CA certificate and key, using the \texttt{sha256WithRSAEncryption} signature algorithm.
- SSL certificates use these Common Name (CN) values, with the appropriate certificate type (CA, Server, Client):

  - \texttt{ca.pem}: MySQL\_Server\_suffix\_Auto\_Generated\_CA\_Certificate
  - \texttt{server-cert.pem}: MySQL\_Server\_suffix\_Auto\_Generated\_Server\_Certificate
  - \texttt{client-cert.pem}: MySQL\_Server\_suffix\_Auto\_Generated\_Client\_Certificate

The \texttt{suffix} value is based on the MySQL version number. For files generated by \texttt{mysql\_ssl\_rsa\_setup}, the suffix can be specified explicitly using the \texttt{--suffix} option.

For files generated by the server, if the resulting CN values exceed 64 characters, the \texttt{_suffix} portion of the name is omitted.

- SSL files have blank values for Country (C), State or Province (ST), Organization (O), Organization Unit Name (OU) and email address.
- SSL files created by the server or by \texttt{mysql\_ssl\_rsa\_setup} are valid for ten years from the time of generation.
- RSA files do not expire.
- SSL files have different serial numbers for each certificate/key pair (1 for CA, 2 for Server, 3 for Client).
- Files created automatically by the server are owned by the account that runs the server. Files created using \texttt{mysql\_ssl\_rsa\_setup} are owned by the user who invoked that program. This can be changed on systems that support the \texttt{chown()} system call if the program is invoked by \texttt{root} and the \texttt{--uid} option is given to specify the user who should own the files.
- On Unix and Unix-like systems, the file access mode is 644 for certificate files (that is, world readable) and 600 for key files (that is, accessible only by the account that runs the server).

To see the contents of an SSL certificate (for example, to check the range of dates over which it is valid), invoke \texttt{openssl} directly:

```
openssl x509 -text -in ca.pem
openssl x509 -text -in server-cert.pem
openssl x509 -text -in client-cert.pem
```

It is also possible to check SSL certificate expiration information using this SQL statement:

```
mysql> SHOW STATUS LIKE 'Ssl\_server\_not%';
+-----------------------+--------------------------+
| Variable_name         | Value                    |
+-----------------------+--------------------------+
```
5.3.2 Creating SSL Certificates and Keys Using openssl

This section describes how to use the openssl command to set up SSL certificate and key files for use by MySQL servers and clients. The first example shows a simplified procedure such as you might use from the command line. The second shows a script that contains more detail. The first two examples are intended for use on Unix and both use the openssl command that is part of OpenSSL. The third example describes how to set up SSL files on Windows.

Note

There are easier alternatives to generating the files required for SSL than the procedure described here: Let the server autogenerate them or use the mysql_ssl_rsa_setup program. See Section 5.3.1, “Creating SSL and RSA Certificates and Keys using MySQL”.

Important

Whatever method you use to generate the certificate and key files, the Common Name value used for the server and client certificates/keys must each differ from the Common Name value used for the CA certificate. Otherwise, the certificate and key files will not work for servers compiled using OpenSSL. A typical error in this case is:

```
ERROR 2026 (HY000): SSL connection error:
error:00000001:lib(0):func(0):reason(1)
```

• Example 1: Creating SSL Files from the Command Line on Unix
• Example 2: Creating SSL Files Using a Script on Unix
• Example 3: Creating SSL Files on Windows

Example 1: Creating SSL Files from the Command Line on Unix

The following example shows a set of commands to create MySQL server and client certificate and key files. You will need to respond to several prompts by the openssl commands. To generate test files, you can press Enter to all prompts. To generate files for production use, you should provide nonempty responses.

```
# Create clean environment
rm -rf newcerts
mkdir newcerts && cd newcerts
# Create CA certificate
openssl genrsa 2048 > ca-key.pem
openssl req -new -x509 -nodes -days 3600
  -key ca-key.pem -out ca.pem
# Create server certificate, remove passphrase, and sign it
# server-cert.pem = public key, server-key.pem = private key
openssl req -newkey rsa:2048 -days 3600
  -nodes -keyout server-key.pem -out server-req.pem
openssl rsa -in server-key.pem -out server-key.pem
openssl x509 -req -in server-req.pem -days 3600
  -CA ca.pem -CAkey ca-key.pem -set_serial 01 -out server-cert.pem
# Create client certificate, remove passphrase, and sign it
# client-cert.pem = public key, client-key.pem = private key
openssl req -newkey rsa:2048 -days 3600
  -nodes -keyout client-key.pem -out client-req.pem
openssl rsa -in client-key.pem -out client-key.pem
openssl x509 -req -in client-req.pem -days 3600
  -CA ca.pem -CAkey ca-key.pem -set_serial 01 -out client-cert.pem
```

After generating the certificates, verify them:
Creating SSL Certificates and Keys Using openssl

```bash
openssl verify -CAfile ca.pem server-cert.pem client-cert.pem
```

You should see a response like this:

```
server-cert.pem: OK
client-cert.pem: OK
```

To see the contents of a certificate (for example, to check the range of dates over which a certificate is valid), invoke `openssl` like this:

```bash
openssl x509 -text -in ca.pem
openssl x509 -text -in server-cert.pem
openssl x509 -text -in client-cert.pem
```

Now you have a set of files that can be used as follows:

- **ca.pem**: Use this to set the `ssl_ca` system variable on the server side and the `--ssl-ca` option on the client side. (The CA certificate, if used, must be the same on both sides.)

- **server-cert.pem, server-key.pem**: Use these to set the `ssl_cert` and `ssl_key` system variables on the server side.

- **client-cert.pem, client-key.pem**: Use these as the arguments to the `--ssl-cert` and `--ssl-key` options on the client side.

For additional usage instructions, see Section 5.1, "Configuring MySQL to Use Encrypted Connections".

### Example 2: Creating SSL Files Using a Script on Unix

Here is an example script that shows how to set up SSL certificate and key files for MySQL. After executing the script, use the files for SSL connections as described in Section 5.1, "Configuring MySQL to Use Encrypted Connections".

```bash
DIR=`pwd`/openssl
PRIV=$DIR/private
mkdir $DIR $PRIV $DIR/newcerts
cp /usr/share/ssl/openssl.cnf $DIR
replace ./demoCA $DIR -- $DIR/openssl.cnf
# Create necessary files: $database, $serial and $new_certs_dir
directory (optional)
touch $DIR/index.txt
echo "01" > $DIR/serial

# Generation of Certificate Authority (CA)

# openssl req -new -x509 -keyout $PRIV/cakey.pem -out $DIR/ca.pem 
# -days 3600 -config $DIR/openssl.cnf
# Sample output:
# Using configuration from /home/jones/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# ................++++++
# .........++++++
# writing new private key to '/home/jones/openssl/private/cakey.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# ----- You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.' the field will be left blank.
# ----- Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
```

136
Creating SSL Certificates and Keys Using openssl

```plaintext
# Common Name (eg, YOUR name) []: MySQL admin
# Email Address []:
# Create server request and key
openssl req -new -keyout $DIR/server-key.pem -out $DIR/server-req.pem -days 3600 -config $DIR/openssl.cnf
# Sample output:
# Using configuration from /home/jones/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# ++++++.........+++++
# writing new private key to '/home/jones/openssl/server-key.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# ----- You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# ----
# Country Name (2 letter code) [AU]: FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]: MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []: MySQL server
# Email Address []:
# Please enter the following 'extra' attributes to be sent with your certificate request
# A challenge password []:
# An optional company name []:
# Remove the passphrase from the key
openssl rsa -in $DIR/server-key.pem -out $DIR/server-key.pem
# Sign server cert
openssl ca -cert $DIR/ca.pem -policy policy_anything 
 -out $DIR/server-cert.pem -config $DIR/openssl.cnf 
 -infiles $DIR/server-req.pem
# Sample output:
# Using configuration from /home/jones/openssl/openssl.cnf
# Enter PEM pass phrase:
# Check that the request matches the signature
# Signature ok
# The Subjects Distinguished Name is as follows
# countryName :PRINTABLE:'FI'
# organizationName :PRINTABLE:'MySQL AB'
# commonName :PRINTABLE:'MySQL admin'
# Certificate is to be certified until Sep 13 14:22:46 2003 GMT
# (365 days)
# Sign the certificate? [y/n]: y
# # 1 out of 1 certificate requests certified, commit? [y/n] y
# Write out database with 1 new entries
# Data Base Updated
# Create client request and key
openssl req -new -keyout $DIR/client-key.pem -out $DIR/client-req.pem -days 3600 -config $DIR/openssl.cnf
# Sample output:
# Using configuration from /home/jones/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# ++++++........................................+++++
```

137
Creating SSL Certificates and Keys Using openssl

```bash
# writing new private key to '/home/jones/openssl/client-key.pem'
Enter PEM pass phrase:
Verifying password - Enter PEM pass phrase:

You are about to be asked to enter information that will be incorporated into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:FI
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []:MySQL user
Email Address []:

Please enter the following 'extra' attributes to be sent with your certificate request
A challenge password []:
An optional company name []:

Remove the passphrase from the key

openssl rsa -in $DIR/client-key.pem -out $DIR/client-key.pem

# Sign client cert
openssl ca -cert $DIR/ca.pem -policy policy_anything
    -out $DIR/client-cert.pem -config $DIR/openssl.cnf
    -infiles $DIR/client-req.pem

Sample output:
Using configuration from /home/jones/openssl/openssl.cnf
Enter PEM pass phrase:
Check that the request matches the signature
Signature ok
The Subjects Distinguished Name is as follows
countryName :PRINTABLE:'FI'
organizationName :PRINTABLE:'MySQL AB'
commonName :PRINTABLE:'MySQL user'
Certificate is to be certified until Sep 13 16:45:17 2003 GMT
(365 days)
Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated

# Create a my.cnf file that you can use to test the certificates
cat <<EOF > $DIR/my.cnf
[client]
ssl-ca=$DIR/ca.pem
ssl-cert=$DIR/client-cert.pem
ssl-key=$DIR/client-key.pem
[mysqld]
ssl_ca=$DIR/ca.pem
ssl_cert=$DIR/server-cert.pem
ssl_key=$DIR/server-key.pem
EOF
```

Example 3: Creating SSL Files on Windows

Download OpenSSL for Windows if it is not installed on your system. An overview of available packages can be seen here:

```
http://www.slproweb.com/products/Win32OpenSSL.html
```
Creating SSL Certificates and Keys Using openssl

Choose the Win32 OpenSSL Light or Win64 OpenSSL Light package, depending on your architecture (32-bit or 64-bit). The default installation location will be C:\OpenSSL-Win32 or C:\OpenSSL-Win64, depending on which package you downloaded. The following instructions assume a default location of C:\OpenSSL-Win32. Modify this as necessary if you are using the 64-bit package.

If a message occurs during setup indicating '...critical component is missing: Microsoft Visual C++ 2008 Redistributables', cancel the setup and download one of the following packages as well, again depending on your architecture (32-bit or 64-bit):

• Visual C++ 2008 Redistributables (x86), available at:


• Visual C++ 2008 Redistributables (x64), available at:


After installing the additional package, restart the OpenSSL setup procedure.

During installation, leave the default C:\OpenSSL-Win32 as the install path, and also leave the default option 'Copy OpenSSL DLL files to the Windows system directory' selected.

When the installation has finished, add C:\OpenSSL-Win32\bin to the Windows System Path variable of your server (depending on your version of Windows, the following path-setting instructions might differ slightly):

1. On the Windows desktop, right-click the My Computer icon, and select Properties.

2. Select the Advanced tab from the System Properties menu that appears, and click the Environment Variables button.

3. Under System Variables, select Path, then click the Edit button. The Edit System Variable dialogue should appear.

4. Add ';C:\OpenSSL-Win32\bin' to the end (notice the semicolon).

5. Press OK 3 times.

6. Check that OpenSSL was correctly integrated into the Path variable by opening a new command console (Start>Run>cmd.exe) and verifying that OpenSSL is available:

Microsoft Windows [Version ...
Copyright (c) 2006 Microsoft Corporation. All rights reserved...
C:\Windows\system32> cd \\
C:\>openssl
OpenSSL> exit <<< If you see the OpenSSL prompt, installation was successful.
C:\>

After OpenSSL has been installed, use instructions similar to those from Example 1 (shown earlier in this section), with the following changes:

• Change the following Unix commands:

```
# Create clean environment
rm -rf newcerts
mkdir newcerts && cd newcerts
```

On Windows, use these commands instead:

```
# Create clean environment
md c:\newcerts
cd c:\newcerts
```

• When a '\' character is shown at the end of a command line, this '\' character must be removed and the command lines entered all on a single line.
After generating the certificate and key files, to use them for SSL connections, see Section 5.1, “Configuring MySQL to Use Encrypted Connections”.

5.3.3 Creating RSA Keys Using openssl

This section describes how to use the openssl command to set up the RSA key files that enable MySQL to support secure password exchange over unencrypted connections for accounts authenticated by the sha256_password and caching_sha2_password plugins.

Note

There are easier alternatives to generating the files required for RSA than the procedure described here: Let the server autogenerate them or use the mysql_ssl_rsa_setup program. See Section 5.3.1, “Creating SSL and RSA Certificates and Keys using MySQL”.

To create the RSA private and public key-pair files, run these commands while logged into the system account used to run the MySQL server so the files will be owned by that account:

```
openssl genrsa -out private_key.pem 2048
openssl rsa -in private_key.pem -pubout -out public_key.pem
```

Those commands create 2,048-bit keys. To create stronger keys, use a larger value.

Then set the access modes for the key files. The private key should be readable only by the server, whereas the public key can be freely distributed to client users:

```
chmod 400 private_key.pem
chmod 444 public_key.pem
```

5.4 Connecting to MySQL Remotely from Windows with SSH

This section describes how to get an encrypted connection to a remote MySQL server with SSH. The information was provided by David Carlson <dcarlson@mplcomm.com>.

1. Install an SSH client on your Windows machine. For a comparison of SSH clients, see http://en.wikipedia.org/wiki/Comparison_of_SSH_clients.

2. Start your Windows SSH client. Set Host_Name = yourmysqlserver_URL_or_IP. Set userid=your_userid to log in to your server. This userid value might not be the same as the user name of your MySQL account.

3. Set up port forwarding. Either do a remote forward (Set local_port: 3306, remote_host: yourmysqlservername_or_ip, remote_port: 3306) or a local forward (Set port: 3306, host: localhost, remote port: 3306).

4. Save everything, otherwise you will have to redo it the next time.

5. Log in to your server with the SSH session you just created.

6. On your Windows machine, start some ODBC application (such as Access).

7. Create a new file in Windows and link to MySQL using the ODBC driver the same way you normally do, except type in localhost for the MySQL host server, not yourmysqlservername.

At this point, you should have an ODBC connection to MySQL, encrypted using SSH.
Chapter 6 Security Components and Plugins

Table of Contents

6.1 Authentication Plugins ............................................................................................................. 142
  6.1.1 Native Pluggable Authentication ......................................................................................... 142
  6.1.2 Caching SHA-2 Pluggable Authentication ......................................................................... 143
  6.1.3 SHA-256 Pluggable Authentication ....................................................................................... 148
  6.1.4 Client-Side Cleartext Pluggable Authentication ................................................................. 152
  6.1.5 PAM Pluggable Authentication ........................................................................................... 153
  6.1.6 Windows Pluggable Authentication ..................................................................................... 163
  6.1.7 LDAP Pluggable Authentication .......................................................................................... 168
  6.1.8 No-Login Pluggable Authentication ..................................................................................... 186
  6.1.9 Socket Peer-Credential Pluggable Authentication ............................................................... 189
  6.1.10 Test Pluggable Authentication .......................................................................................... 191
  6.1.11 Pluggable Authentication System Variables ...................................................................... 193

6.2 The Connection-Control Plugins ............................................................................................ 210
  6.2.1 Connection-Control Plugin Installation ............................................................................. 210
  6.2.2 Connection-Control System and Status Variables ............................................................. 214

6.3 The Password Validation Component ....................................................................................... 216
  6.3.1 Password Validation Component Installation and Uninstallation .................................... 218
  6.3.2 Password Validation Options and Variables ........................................................................ 218
  6.3.3 Transitioning to the Password Validation Component ...................................................... 227

6.4 The MySQL Keyring .................................................................................................................. 228
  6.4.1 Keyring Plugin Installation ................................................................................................. 229
  6.4.2 Using the keyring_file File-Based Plugin ......................................................................... 230
  6.4.3 Using the keyring_encrypted_file Keyring Plugin ............................................................. 231
  6.4.4 Using the keyring_awk KMIP Plugin .................................................................................. 232
  6.4.5 Using the keyring_aws Amazon Web Services Keyring Plugin ....................................... 237
  6.4.6 Using the HashiCorp Vault Keyring Plugin ........................................................................ 240
  6.4.7 Migrating Keys Between Keyring Keystores ..................................................................... 247
  6.4.8 Supported Keyring Key Types and Lengths ....................................................................... 249
  6.4.9 General-Purpose Keyring Key-Management Functions ..................................................... 251
  6.4.10 Plugin-Specific Keyring Key-Management Functions ...................................................... 258
  6.4.11 Keyring Command Options .............................................................................................. 259
  6.4.12 Keyring System Variables ............................................................................................... 261

6.5 MySQL Enterprise Audit .......................................................................................................... 271
  6.5.1 Audit Log Components ..................................................................................................... 272
  6.5.2 Installing or Uninstalling MySQL Enterprise Audit .......................................................... 272
  6.5.3 MySQL Enterprise Audit Security Considerations ............................................................. 274
  6.5.4 Audit Log File Formats ...................................................................................................... 274
  6.5.5 Audit Log Logging Configuration ....................................................................................... 293
  6.5.6 Audit Log Filtering ............................................................................................................. 302
  6.5.7 Writing Audit Log Filter Definitions .................................................................................. 305
  6.5.8 Legacy Mode Audit Log Filtering ....................................................................................... 317
  6.5.9 Audit Log Reference .......................................................................................................... 319
  6.5.10 Audit Log Restrictions .................................................................................................... 337

6.6 The Audit Message Component ............................................................................................... 337

6.7 MySQL Enterprise Firewall ..................................................................................................... 340
  6.7.1 MySQL Enterprise Firewall Components ........................................................................... 341
  6.7.2 Installing or Uninstalling MySQL Enterprise Firewall ....................................................... 341
  6.7.3 Using MySQL Enterprise Firewall ...................................................................................... 344
  6.7.4 MySQL Enterprise Firewall Reference ............................................................................... 348

6.8 MySQL Enterprise Data Masking and De-Identification ......................................................... 352
  6.8.1 MySQL Enterprise Data Masking and De-Identification Components ............................. 354
  6.8.2 Installing or Uninstalling MySQL Enterprise Data Masking and De-Identification ............ 354
MySQL includes several components and plugins that implement security features:

- Plugins for authenticating attempts by clients to connect to MySQL Server. Plugins are available for several authentication protocols. For general discussion of the authentication process, see Section 4.17, “Pluggable Authentication”. For characteristics of specific authentication plugins, see Section 6.1, “Authentication Plugins”.

- A password-validation component for implementing password strength policies and assessing the strength of potential passwords. See Section 6.3, “The Password Validation Component”.

- Keyring plugins that provide secure storage for sensitive information. See Section 6.4, “The MySQL Keyring”.

- (MySQL Enterprise Edition only) MySQL Enterprise Audit, implemented using a server plugin, uses the open MySQL Audit API to enable standard, policy-based monitoring and logging of connection and query activity executed on specific MySQL servers. Designed to meet the Oracle audit specification, MySQL Enterprise Audit provides an out of box, easy to use auditing and compliance solution for applications that are governed by both internal and external regulatory guidelines. See Section 6.5, “MySQL Enterprise Audit”.

- A user-defined function enables applications to add their own message events to the audit log. See Section 6.6, “The Audit Message Component”.

- (MySQL Enterprise Edition only) MySQL Enterprise Firewall, an application-level firewall that enables database administrators to permit or deny SQL statement execution based on matching against whitelists of accepted statement patterns. This helps harden MySQL Server against attacks such as SQL injection or attempts to exploit applications by using them outside of their legitimate query workload characteristics. See Section 6.7, “MySQL Enterprise Firewall”.

- (MySQL Enterprise Edition only) MySQL Enterprise Data Masking and De-Identification, implemented as a plugin library containing a plugin and a set of user-defined functions. Data masking hides sensitive information by replacing real values with substitutes. MySQL Enterprise Data Masking and De-Identification functions enable masking existing data using several methods such as obfuscation (removing identifying characteristics), generation of formatted random data, and data replacement or substitution. See Section 6.8, “MySQL Enterprise Data Masking and De-Identification”.

### 6.1 Authentication Plugins

The following sections describe pluggable authentication methods available in MySQL and the plugins that implement these methods. For general discussion of the authentication process, see Section 4.17, “Pluggable Authentication”.

The default plugin is indicated by the value of the `default_authentication_plugin` system variable.

#### 6.1.1 Native Pluggable Authentication

MySQL includes a `mysql_native_password` plugin that implements native authentication; that is, authentication based on the password hashing method in use from before the introduction of pluggable authentication.

The following table shows the plugin names on the server and client sides.
Table 6.1 Plugin and Library Names for Native Password Authentication

<table>
<thead>
<tr>
<th>Plugin or File</th>
<th>Plugin or File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server-side plugin</td>
<td>mysql_native_password</td>
</tr>
<tr>
<td>Client-side plugin</td>
<td>mysql_native_password</td>
</tr>
<tr>
<td>Library file</td>
<td>None (plugins are built in)</td>
</tr>
</tbody>
</table>

The following sections provide installation and usage information specific to native pluggable authentication:

- Installing Native Pluggable Authentication
- Using Native Pluggable Authentication

For general information about pluggable authentication in MySQL, see Section 4.17, “Pluggable Authentication”.

### Installing Native Pluggable Authentication

The `mysql_native_password` plugin exists in server and client forms:

- The server-side plugin is built into the server, need not be loaded explicitly, and cannot be disabled by unloading it.
- The client-side plugin is built into the `libmysqlclient` client library and is available to any program linked against `libmysqlclient`.

### Using Native Pluggable Authentication

MySQL client programs use `mysql_native_password` by default. The `--default-auth` option can be used as a hint about which client-side plugin the program can expect to use:

```
shell> mysql --default-auth=mysql_native_password ...
```

### 6.1.2 Caching SHA-2 Pluggable Authentication

MySQL provides two authentication plugins that implement SHA-256 hashing for user account passwords:

- `sha256_password`: Implements basic SHA-256 authentication.
- `caching_sha2_password`: Implements SHA-256 authentication (like `sha256_password`), but uses caching on the server side for better performance and has additional features for wider applicability.

This section describes the caching SHA-2 authentication plugin. For information about the original basic (noncaching) plugin, see Section 6.1.3, “SHA-256 Pluggable Authentication”.

**Important**

In MySQL 8.0, `caching_sha2_password` is the default authentication plugin rather than `mysql_native_password`. For information about the implications of this change for server operation and compatibility of the server with clients and connectors, see `caching_sha2_password` as the Preferred Authentication Plugin.

**Important**

To connect to the server using an account that authenticates with the `caching_sha2_password` plugin, you must use either a secure connection or an unencrypted connection that supports password exchange using...
an RSA key pair, as described later in this section. Either way, the caching_sha2_password plugin uses MySQL’s encryption capabilities. See Chapter 5, Using Encrypted Connections.

Note
In the name sha256_password, “sha256” refers to the 256-bit digest length the plugin uses for encryption. In the name caching_sha2_password, “sha2” refers more generally to the SHA-2 class of encryption algorithms, of which 256-bit encryption is one instance. The latter name choice leaves room for future expansion of possible digest lengths without changing the plugin name.

The caching_sha2_password plugin has these advantages, compared to sha256_password:

• On the server side, an in-memory cache enables faster reauthentication of users who have connected previously when they connect again.

• RSA-based password exchange is available regardless of the SSL library against which MySQL is linked.

• Support is provided for client connections that use the Unix socket-file and shared-memory protocols.

The following table shows the plugin names on the server and client sides.

Table 6.2 Plugin and Library Names for SHA-2 Authentication

<table>
<thead>
<tr>
<th>Plugin or File</th>
<th>Plugin or File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server-side plugin</td>
<td>caching_sha2_password</td>
</tr>
<tr>
<td>Client-side plugin</td>
<td>caching_sha2_password</td>
</tr>
<tr>
<td>Library file</td>
<td>None (plugins are built in)</td>
</tr>
</tbody>
</table>

The following sections provide installation and usage information specific to caching SHA-2 pluggable authentication:

• Installing SHA-2 Pluggable Authentication

• Using SHA-2 Pluggable Authentication

• Cache Operation for SHA-2 Pluggable Authentication

For general information about pluggable authentication in MySQL, see Section 4.17, “Pluggable Authentication”.

Installing SHA-2 Pluggable Authentication

The caching_sha2_password plugin exists in server and client forms:

• The server-side plugin is built into the server, need not be loaded explicitly, and cannot be disabled by unloading it.

• The client-side plugin is built into the libmysqlclient client library and is available to any program linked against libmysqlclient.

The server-side plugin uses the sha2_cache_cleaner audit plugin as a helper to perform password cache management. sha2_cache_cleaner, like caching_sha2_password, is built in and need not be installed.

Using SHA-2 Pluggable Authentication

To set up an account that uses the caching_sha2_password plugin for SHA-256 password hashing, use the following statement, where password is the desired account password:
CREATE USER 'sha2user'@'localhost'
IDENTIFIED WITH caching_sha2_password BY 'password';

The server assigns the `caching_sha2_password` plugin to the account and uses it to encrypt the password using SHA-256, storing those values in the `plugin` and `authentication_string` columns of the `mysql.user` system table.

The preceding instructions do not assume that `caching_sha2_password` is the default authentication plugin. If `caching_sha2_password` is the default authentication plugin, a simpler `CREATE USER` syntax can be used.

To start the server with the default authentication plugin set to `caching_sha2_password`, put these lines in the server option file:

```
[mysqld]
default_authentication_plugin=caching_sha2_password
```

That causes the `caching_sha2_password` plugin to be used by default for new accounts. As a result, it is possible to create the account and set its password without naming the plugin explicitly:

```
CREATE USER 'sha2user'@'localhost' IDENTIFIED BY 'password';
```

Another consequence of setting `default_authentication_plugin` to `caching_sha2_password` is that, to use some other plugin for account creation, you must specify that plugin explicitly. For example, to use the `mysql_native_password` plugin, use this statement:

```
CREATE USER 'nativeuser'@'localhost'
IDENTIFIED WITH mysql_native_password BY 'password';
```

`caching_sha2_password` supports connections over secure transport. If you follow the RSA configuration procedure given later in this section, it also supports encrypted password exchange using RSA over unencrypted connections. RSA support has these characteristics:

- On the server side, two system variables name the RSA private and public key-pair files: `caching_sha2_password_private_key_path` and `caching_sha2_password_public_key_path`. The database administrator must set these variables at server startup if the key files to use have names that differ from the system variable default values.

- The server uses the `caching_sha2_password_auto_generate_rsa_keys` system variable to determine whether to automatically generate the RSA key-pair files. See [Section 5.3, “Creating SSL and RSA Certificates and Keys”](#).

- The `Caching_sha2_password_rsa_public_key` status variable displays the RSA public key value used by the `caching_sha2_password` authentication plugin.

- Clients that are in possession of the RSA public key can perform RSA key pair-based password exchange with the server during the connection process, as described later.

- For connections by accounts that authenticate with `caching_sha2_password` and RSA key pair-based password exchange, the server does not send the RSA public key to clients by default. Clients can use a client-side copy of the required public key, or request the public key from the server.

Use of a trusted local copy of the public key enables the client to avoid a round trip in the client/server protocol, and is more secure than requesting the public key from the server. On the other hand, requesting the public key from the server is more convenient (it requires no management of a client-side file) and may be acceptable in secure network environments.

- For command-line clients, use the `--server-public-key-path` option to specify the RSA public key file. Use the `--get-server-public-key` option to request the public key from the server. The following programs support the two options: `mysql`, `mysqlsh`, `mysqladmin`, `mysqlbinlog`, `mysqlcheck`, `mysqldump`, `mysqlimport`, `mysqlpump`, `mysqlshow`, `mysqlslap`, `mysqltest`, `mysql_upgrade`. 
For programs that use the C API, call `mysql_options()` to specify the RSA public key file by passing the `MYSQL_SERVER_PUBLIC_KEY` option and the name of the file, or request the public key from the server by passing the `MYSQL_OPT_GET_SERVER_PUBLIC_KEY` option.

For replication slaves, use the `CHANGE MASTER TO` statement with the `MASTER_PUBLIC_KEY_PATH` option to specify the RSA public key file, or the `GET_MASTER_PUBLIC_KEY` option to request the public key from the master. For Group Replication, the `group_replication_recovery_public_key_path` and `group_replication_recovery_get_public_key` system variables serve the same purpose.

In all cases, if the option is given to specify a valid public key file, it takes precedence over the option to request the public key from the server.

For clients that use the `caching_sha2_password` plugin, passwords are never exposed as cleartext when connecting to the server. How password transmission occurs depends on whether a secure connection or RSA encryption is used:

- If the connection is secure, an RSA key pair is unnecessary and is not used. This applies to encrypted TCP connections that use TLS, as well as Unix socket-file and shared-memory connections. The password is sent as cleartext but cannot be snooped because the connection is secure.

- If the connection is not secure, an RSA key pair is used. This applies to unencrypted TCP connections without TLS and named-pipe connections. RSA is used only for password exchange between client and server, to prevent password snooping. When the server receives the encrypted password, it decrypts it. A scramble is used in the encryption to prevent repeat attacks.

To enable use of an RSA key pair for password exchange during the client connection process, use the following procedure:

1. Create the RSA private and public key-pair files using the instructions in Section 5.3, “Creating SSL and RSA Certificates and Keys”.

2. If the private and public key files are located in the data directory and are named `private_key.pem` and `public_key.pem` (the default values of the `caching_sha2_password_private_key_path` and `caching_sha2_password_public_key_path` system variables), the server uses them automatically at startup.

   Otherwise, to name the key files explicitly, set the system variables to the key file names in the server option file. If the files are located in the server data directory, you need not specify their full path names:

   ```sql
   [mysqld]
   caching_sha2_password_private_key_path=myprivkey.pem
   caching_sha2_password_public_key_path=mypubkey.pem
   ```

   If the key files are not located in the data directory, or to make their locations explicit in the system variable values, use full path names:

   ```sql
   [mysqld]
   caching_sha2_password_private_key_path=/usr/local/mysql/myprivkey.pem
   caching_sha2_password_public_key_path=/usr/local/mysql/mypubkey.pem
   ```

3. Restart the server, then connect to it and check the `Caching_sha2_password_rsa_public_key` status variable value. The value will differ from that shown here, but should be nonempty:

   ```sql
   mysql> SHOW STATUS LIKE 'Caching_sha2_password_rsa_public_key'\G
   *************************** 1. row ***************************
   Variable_name: Caching_sha2_password_rsa_public_key
   Value: -----BEGIN PUBLIC KEY-----
If the value is empty, the server found some problem with the key files. Check the error log for diagnostic information.

After the server has been configured with the RSA key files, accounts that authenticate with the `caching_sha2_password` plugin have the option of using those key files to connect to the server. As mentioned previously, such accounts can use either a secure connection (in which case RSA is not used) or an unencrypted connection that performs password exchange using RSA. Suppose that an unencrypted connection is used. For example:

```
shell> mysql --ssl-mode=DISABLED -u sha2user -p
Enter password: password
```

For this connection attempt by `sha2user`, the server determines that `caching_sha2_password` is the appropriate authentication plugin and invokes it (because that was the plugin specified at `CREATE USER` time). The plugin finds that the connection is not encrypted and thus requires the password to be transmitted using RSA encryption. However, the server does not send the public key to the client, and the client provided no public key, so it cannot encrypt the password and the connection fails:

```
ERROR 2061 (HY000): Authentication plugin 'caching_sha2_password' reported error: Authentication requires secure connection.
```

To request the RSA public key from the server, specify the `--get-server-public-key` option:

```
shell> mysql --ssl-mode=DISABLED -u sha2user -p --get-server-public-key
Enter password: password
```

In this case, the server sends the RSA public key to the client, which uses it to encrypt the password and returns the result to the server. The plugin uses the RSA private key on the server side to decrypt the password and accepts or rejects the connection based on whether the password is correct.

Alternatively, if the client has a file containing a local copy of the RSA public key required by the server, it can specify the file using the `--server-public-key-path` option:

```
shell> mysql --ssl-mode=DISABLED -u sha2user -p --server-public-key-path=file_name
Enter password: password
```

In this case, the client uses the public key to encrypt the password and returns the result to the server. The plugin uses the RSA private key on the server side to decrypt the password and accepts or rejects the connection based on whether the password is correct.

The public key value in the file named by the `--server-public-key-path` option should be the same as the key value in the server-side file named by the `caching_sha2_password_public_key_path` system variable. If the key file contains a valid public key value but the value is incorrect, an access-denied error occurs. If the key file does not contain a valid public key, the client program cannot use it.

Client users can obtain the RSA public key two ways:

- The database administrator can provide a copy of the public key file.
- A client user who can connect to the server some other way can use a `SHOW STATUS LIKE 'Caching_sha2_password_rsa_public_key'` statement and save the returned key value in a file.

**Cache Operation for SHA-2 Pluggable Authentication**

On the server side, the `caching_sha2_password` plugin uses an in-memory cache for faster authentication of clients who have connected previously. Entries consist of account-name/password-hash pairs. The cache works like this:
1. When a client connects, `caching_sha2_password` checks whether the client and password match some cache entry. If so, authentication succeeds.

2. If there is no matching cache entry, the plugin attempts to verify the client against the credentials in the `mysql.user` system table. If this succeeds, `caching_sha2_password` adds an entry for the client to the hash. Otherwise, authentication fails and the connection is rejected.

In this way, when a client first connects, authentication against the `mysql.user` system table occurs. When the client connects subsequently, faster authentication against the cache occurs.

Password cache operations other than adding entries are handled by the `sha2_cache_cleaner` audit plugin, which performs these actions on behalf of `caching_sha2_password`:

- It clears the cache entry for any account that is renamed or dropped, or any account for which the credentials or authentication plugin are changed.
- It empties the cache when the `FLUSH PRIVILEGES` statement is executed.
- It empties the cache at server shutdown. (This means the cache is not persistent across server restarts.)

Cache clearing operations affect the authentication requirements for subsequent client connections. For each user account, the first client connection for the user after any of the following operations must use a secure connection (made using TCP using TLS credentials, a Unix socket file, or shared memory) or RSA key pair-based password exchange:

- After account creation.
- After a password change for the account.
- After `RENAME USER` for the account.
- After `FLUSH PRIVILEGES`.

`FLUSH PRIVILEGES` clears the entire cache and affects all accounts that use the `caching_sha2_password` plugin. The other operations clear specific cache entries and affect only accounts that are part of the operation.

Once the user authenticates successfully, the account is entered into the cache and subsequent connections do not require a secure connection or the RSA key pair, until another cache clearing event occurs that affects the account. (When the cache can be used, the server uses a challenge-response mechanism that does not use cleartext password transmission and does not require a secure connection.)

### 6.1.3 SHA-256 Pluggable Authentication

MySQL provides two authentication plugins that implement SHA-256 hashing for user account passwords:

- `sha256_password`: Implements basic SHA-256 authentication.
- `caching_sha2_password`: Implements SHA-256 authentication (like `sha256_password`), but uses caching on the server side for better performance and has additional features for wider applicability.

This section describes the original noncaching SHA-2 authentication plugin. For information about the caching plugin, see Section 6.1.2, “Caching SHA-2 Pluggable Authentication”.

**Important**

In MySQL 8.0, `caching_sha2_password` is the default authentication plugin rather than `mysql_native_password`. For information about the implications
SHA-256 Pluggable Authentication

Because `caching_sha2_password` is the default authentication plugin in MySQL 8.0 and provides a superset of the capabilities of the `sha256_password` authentication plugin, `sha256_password` is deprecated and will be removed in a future MySQL version. MySQL accounts that authenticate using `sha256_password` should be migrated to use `caching_sha2_password` instead.

Important

To connect to the server using an account that authenticates with the `sha256_password` plugin, you must use either a TLS connection or an unencrypted connection that supports password exchange using an RSA key pair, as described later in this section. Either way, the `sha256_password` plugin uses MySQL's encryption capabilities. See Chapter 5, Using Encrypted Connections.

Note

In the name `sha256_password`, “sha256” refers to the 256-bit digest length the plugin uses for encryption. In the name `caching_sha2_password`, “sha2” refers more generally to the SHA-2 class of encryption algorithms, of which 256-bit encryption is one instance. The latter name choice leaves room for future expansion of possible digest lengths without changing the plugin name.

The following table shows the plugin names on the server and client sides.

Table 6.3 Plugin and Library Names for SHA-256 Authentication

<table>
<thead>
<tr>
<th>Plugin or File</th>
<th>Plugin or File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server-side plugin</td>
<td>sha256_password</td>
</tr>
<tr>
<td>Client-side plugin</td>
<td>sha256_password</td>
</tr>
<tr>
<td>Library file</td>
<td>None (plugins are built in)</td>
</tr>
</tbody>
</table>

The following sections provide installation and usage information specific to SHA-256 pluggable authentication:

- Installing SHA-256 Pluggable Authentication
- Using SHA-256 Pluggable Authentication

For general information about pluggable authentication in MySQL, see Section 4.17, “Pluggable Authentication”.

Installing SHA-256 Pluggable Authentication

The `sha256_password` plugin exists in server and client forms:

- The server-side plugin is built into the server, need not be loaded explicitly, and cannot be disabled by unloading it.
- The client-side plugin is built into the `libmysqlclient` client library and is available to any program linked against `libmysqlclient`.

Using SHA-256 Pluggable Authentication

To set up an account that uses the `sha256_password` plugin for SHA-256 password hashing, use the following statement, where `password` is the desired account password:
CREATE USER 'sha256user'@'localhost'
IDENTIFIED WITH sha256_password BY 'password';

The server assigns the sha256_password plugin to the account and uses it to encrypt the password using SHA-256, storing those values in the plugin and authentication_string columns of the mysql.user system table.

The preceding instructions do not assume that sha256_password is the default authentication plugin. If sha256_password is the default authentication plugin, a simpler CREATE USER syntax can be used.

To start the server with the default authentication plugin set to sha256_password, put these lines in the server option file:

```
[mysqld]
default_authentication_plugin=sha256_password
```

That causes the sha256_password plugin to be used by default for new accounts. As a result, it is possible to create the account and set its password without naming the plugin explicitly:

```
CREATE USER 'sha256user'@'localhost' IDENTIFIED BY 'password';
```

Another consequence of setting default_authentication_plugin to sha256_password is that, to use some other plugin for account creation, you must specify that plugin explicitly. For example, to use the mysql_native_password plugin, use this statement:

```
CREATE USER 'nativeuser'@'localhost'
IDENTIFIED WITH mysql_native_password BY 'password';
```

sha256_password supports connections over secure transport. sha256_password also supports encrypted password exchange using RSA over unencrypted connections if MySQL is compiled using OpenSSL, and the MySQL server to which you wish to connect is configured to support RSA (using the RSA configuration procedure given later in this section).

RSA support has these characteristics:

- On the server side, two system variables name the RSA private and public key-pair files: sha256_password_private_key_path and sha256_password_public_key_path. The database administrator must set these variables at server startup if the key files to use have names that differ from the system variable default values.
- The server uses the sha256_password_auto_generate_rsa_keys system variable to determine whether to automatically generate the RSA key-pair files. See Section 5.3, "Creating SSL and RSA Certificates and Keys".
- The Rsa_public_key status variable displays the RSA public key value used by the sha256_password authentication plugin.
- Clients that are in possession of the RSA public key can perform RSA key pair-based password exchange with the server during the connection process, as described later.
- For connections by accounts that authenticate with sha256_password and RSA public key pair-based password exchange, the server sends the RSA public key to the client as needed. However, if a copy of the public key is available on the client host, the client can use it to save a round trip in the client/server protocol:
  - For these command-line clients, use the --server-public-key-path option to specify the RSA public key file: mysql, mysqladmin, mysqlbinlog, mysqlcheck, mysqldump, mysqlimport, mysqlpump, mysqlshow, mysqlslap, mysqltest, mysql_upgrade.
  - For programs that use the C API, call mysql_options() to specify the RSA public key file by passing the MYSQL_SERVER_PUBLIC_KEY option and the name of the file.
For replication slaves, use the `CHANGE MASTER TO` statement with the `MASTER_PUBLIC_KEY_PATH` option to specify the RSA public key file. For Group Replication, the `group_replication_recovery_get_public_key` system variable serves the same purpose.

For clients that use the `sha256_password` plugin, passwords are never exposed as cleartext when connecting to the server. How password transmission occurs depends on whether a secure connection or RSA encryption is used:

- If the connection is secure, an RSA key pair is unnecessary and is not used. This applies to encrypted connections that use TLS. The password is sent as cleartext but cannot be snooped because the connection is secure.

- If the connection is not secure, and an RSA key pair is available, the connection remains unencrypted. This applies to unencrypted connections without TLS. RSA is used only for password exchange between client and server, to prevent password snooping. When the server receives the encrypted password, it decrypts it. A scramble is used in the encryption to prevent repeat attacks.

- If a secure connection is not used and RSA encryption is not available, the connection attempt fails because the password cannot be sent without being exposed as cleartext.

Note

To use RSA password encryption with `sha256_password`, the client and server both must be compiled using OpenSSL, not just one of them.

Assuming that MySQL has been compiled using OpenSSL, use the following procedure to enable use of an RSA key pair for password exchange during the client connection process:

1. Create the RSA private and public key-pair files using the instructions in Section 5.3, “Creating SSL and RSA Certificates and Keys”.

2. If the private and public key files are located in the data directory and are named `private_key.pem` and `public_key.pem` (the default values of the `sha256_password_private_key_path` and `sha256_password_public_key_path` system variables), the server uses them automatically at startup. Otherwise, to name the key files explicitly, set the system variables to the key file names in the server option file. If the files are located in the server data directory, you need not specify their full path names:

   ```plaintext
   [mysqld]
   sha256_password_private_key_path=myprivkey.pem
   sha256_password_public_key_path=mypubkey.pem
   ```

   If the key files are not located in the data directory, or to make their locations explicit in the system variable values, use full path names:

   ```plaintext
   [mysqld]
   sha256_password_private_key_path=/usr/local/mysql/myprivkey.pem
   sha256_password_public_key_path=/usr/local/mysql/mypubkey.pem
   ```

3. Restart the server, then connect to it and check the `Rsa_public_key` status variable value. The value will differ from that shown here, but should be nonempty:

   ```plaintext
   mysql> SHOW STATUS LIKE 'Rsa_public_key' \G
   *************************** 1. row ***************************
   Variable_name: Rsa_public_key
   Value: -----BEGIN PUBLIC KEY-----
   MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDO9nRUDd+KvSZgY7cNB2KgWcEJ77J
   aLgJOBCIDmNVnyU54OT/11c2sxiyfaDMe8fCJ642WtnKbY2gkt1IMjUB5ogd5kJ
   g8aV7EktKwyhB0c3QIDAQAAB
   -----END PUBLIC KEY-----
   ```
If the value is empty, the server found some problem with the key files. Check the error log for diagnostic information.

After the server has been configured with the RSA key files, accounts that authenticate with the sha256_password plugin have the option of using those key files to connect to the server. As mentioned previously, such accounts can use either a secure connection (in which case RSA is not used) or an unencrypted connection that performs password exchange using RSA. Suppose that an unencrypted connection is used. For example:

```
shell> mysql --ssl-mode=DISABLED -u sha256user -p
Enter password: password
```

For this connection attempt by sha256user, the server determines that sha256_password is the appropriate authentication plugin and invokes it (because that was the plugin specified at CREATE USER time). The plugin finds that the connection is not encrypted and thus requires the password to be transmitted using RSA encryption. In this case, the plugin sends the RSA public key to the client, which uses it to encrypt the password and returns the result to the server. The plugin uses the RSA private key on the server side to decrypt the password and accepts or rejects the connection based on whether the password is correct.

The server sends the RSA public key to the client as needed. However, if the client has a file containing a local copy of the RSA public key required by the server, it can specify the file using the --server-public-key-path option:

```
shell> mysql --ssl-mode=DISABLED -u sha256user -p --server-public-key-path=file_name
Enter password: password
```

The public key value in the file named by the --server-public-key-path option should be the same as the key value in the server-side file named by the sha256_password_public_key_path system variable. If the key file contains a valid public key value but the value is incorrect, an access-denied error occurs. If the key file does not contain a valid public key, the client program cannot use it. In this case, the sha256_password plugin sends the public key to the client as if no --server-public-key-path option had been specified.

Client users can obtain the RSA public key two ways:

- The database administrator can provide a copy of the public key file.
- A client user who can connect to the server some other way can use a SHOW STATUS LIKE 'Rsa_public_key' statement and save the returned key value in a file.

### 6.1.4 Client-Side Cleartext Pluggable Authentication

A client-side authentication plugin is available that enables clients to send passwords to the server as cleartext, without hashing or encryption. This plugin is built into the MySQL client library.

The following table shows the plugin name.

<table>
<thead>
<tr>
<th>Plugin or File</th>
<th>Plugin or File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server-side plugin</td>
<td>None, see discussion</td>
</tr>
<tr>
<td>Client-side plugin</td>
<td>mysql_clear_password</td>
</tr>
<tr>
<td>Library file</td>
<td>None (plugin is built in)</td>
</tr>
</tbody>
</table>

Many client-side authentication plugins perform hashing or encryption of a password before the client sends it to the server. This enables clients to avoid sending passwords as cleartext.

Hashing or encryption cannot be done for authentication schemes that require the server to receive the password as entered on the client side. In such cases, the client-side mysql_clear_password plugin is used, which enables the client to send the password to the server as cleartext. There is
PAM Pluggable Authentication

no corresponding server-side plugin. Rather, `mysql_clear_password` can be used on the client side in concert with any server-side plugin that needs a cleartext password. (Examples are the PAM and simple LDAP authentication plugins; see Section 6.1.5, “PAM Pluggable Authentication”, and Section 6.1.7, “LDAP Pluggable Authentication”.)

The following discussion provides usage information specific to cleartext pluggable authentication. For general information about pluggable authentication in MySQL, see Section 4.17, “Pluggable Authentication”.

**Note**

Sending passwords as cleartext may be a security problem in some configurations. To avoid problems if there is any possibility that the password would be intercepted, clients should connect to MySQL Server using a method that protects the password. Possibilities include SSL (see Chapter 5, Using Encrypted Connections), IPsec, or a private network.

To make inadvertent use of the `mysql_clear_password` plugin less likely, MySQL clients must explicitly enable it. This can be done in several ways:

- Set the `LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN` environment variable to a value that begins with 1, Y, or y. This enables the plugin for all client connections.
- The `mysql`, `mysqladmin`, `mysqlcheck`, `mysqldump`, `mysqlshow`, and `mysqlslap` client programs support an `--enable-cleartext-plugin` option that enables the plugin on a per-invocation basis.
- The `mysql_options()` C API function supports a `MYSQL_ENABLE_CLEARTEXT_PLUGIN` option that enables the plugin on a per-connection basis. Also, any program that uses `libmysqlclient` and reads option files can enable the plugin by including an `enable-cleartext-plugin` option in an option group read by the client library.

### 6.1.5 PAM Pluggable Authentication

**Note**

PAM pluggable authentication is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see [https://www.mysql.com/products/](https://www.mysql.com/products/).

MySQL Enterprise Edition supports an authentication method that enables MySQL Server to use PAM (Pluggable Authentication Modules) to authenticate MySQL users. PAM enables a system to use a standard interface to access various kinds of authentication methods, such as traditional Unix passwords or an LDAP directory.

PAM pluggable authentication provides these capabilities:

- **External authentication**: PAM authentication enables MySQL Server to accept connections from users defined outside the MySQL grant tables and that authenticate using methods supported by PAM.
- **Proxy user support**: PAM authentication can return to MySQL a user name different from the external user name passed by the client program, based on the PAM groups the external user is a member of and the authentication string provided. This means that the plugin can return the MySQL user that defines the privileges the external PAM-authenticated user should have. For example, an operating system user named `joe` can connect and have the privileges of a MySQL user named `developer`.

PAM pluggable authentication has been tested on Linux and macOS.

The following table shows the plugin and library file names. The file name suffix might differ on your system. The file must be located in the directory named by the `plugin_dir` system variable. For installation information, see [Installing PAM Pluggable Authentication](https://www.mysql.com/products/).
Table 6.5 Plugin and Library Names for PAM Authentication

<table>
<thead>
<tr>
<th>Plugin or File</th>
<th>Plugin or File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server-side plugin</td>
<td>authentication_pam</td>
</tr>
<tr>
<td>Client-side plugin</td>
<td>mysql_clear_password</td>
</tr>
<tr>
<td>Library file</td>
<td>authentication_pam.so</td>
</tr>
</tbody>
</table>

The client-side `mysql_clear_password` cleartext plugin that communicates with the server-side PAM plugin is built into the `libmysqlclient` client library and is included in all distributions, including community distributions. Inclusion of the client-side cleartext plugin in all MySQL distributions enables clients from any distribution to connect to a server that has the server-side PAM plugin loaded.

The following sections provide installation and usage information specific to PAM pluggable authentication:

- How PAM Authentication of MySQL Users Works
- Installing PAM Pluggable Authentication
- Uninstalling PAM Pluggable Authentication
- Using PAM Pluggable Authentication
- PAM Unix Password Authentication without Proxy Users
- PAM LDAP Authentication without Proxy Users
- PAM Unix Password Authentication with Proxy Users and Group Mapping
- PAM Authentication Access to Unix Password Store
- PAM Authentication Debugging

For general information about pluggable authentication in MySQL, see Section 4.17, “Pluggable Authentication”. For information about the `mysql_clear_password` plugin, see Section 6.1.4, “Client-Side Cleartext Pluggable Authentication”. For proxy user information, see Section 4.18, “Proxy Users”.

How PAM Authentication of MySQL Users Works

This section provides a general overview of how MySQL and PAM work together to authenticate MySQL users. For examples showing how to set up MySQL accounts to use specific PAM services, see Using PAM Pluggable Authentication.

1. The client program and the server communicate, with the client sending to the server the client user name (the operating system user name by default) and password:
   - The client user name is the external user name.
   - For accounts that use the PAM server-side authentication plugin, the corresponding client-side plugin is `mysql_clear_password`. This client-side plugin performs no password hashing, with the result that the client sends the password to the server as cleartext.

2. The server finds a matching MySQL account based on the external user name and the host from which the client connects. The PAM plugin uses the information passed to it by MySQL Server (such as user name, host name, password, and authentication string). When you define a MySQL account that authenticates using PAM, the authentication string contains:
   - A PAM service name, which is a name that the system administrator can use to refer to an authentication method for a particular application. There can be multiple applications associated with a single database server instance, so the choice of service name is left to the SQL application developer.
   - Optionally, if proxying is to be used, a mapping from PAM groups to MySQL user names.
3. The plugin uses the PAM service named in the authentication string to check the user credentials and returns 'Authentication succeeded, Username is user_name' or 'Authentication failed'. The password must be appropriate for the password store used by the PAM service. Examples:

- For traditional Unix passwords, the service looks up passwords stored in the /etc/shadow file.
- For LDAP, the service looks up passwords stored in an LDAP directory.

If the credentials check fails, the server refuses the connection.

4. Otherwise, the authentication string indicates whether proxying occurs. If the string contains no PAM group mapping, proxying does not occur. In this case, the MySQL user name is the same as the external user name.

5. Otherwise, proxying is indicated based on the PAM group mapping, with the MySQL user name determined based on the first matching group in the mapping list. The meaning of “PAM group” depends on the PAM service. Examples:

- For traditional Unix passwords, groups are Unix groups defined in the /etc/group file, possibly supplemented with additional PAM information in a file such as /etc/security/group.conf.
- For LDAP, groups are LDAP groups defined in an LDAP directory.

If the proxy user (the external user) has the PROXY privilege for the proxied MySQL user name, proxying occurs, with the proxy user assuming the privileges of the proxied user.

Installing PAM Pluggable Authentication

This section describes how to install the PAM authentication plugin. For general information about installing plugins, see Installing and Uninstalling Plugins.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the plugin_dir system variable). If necessary, configure the plugin directory location by setting the value of plugin_dir at server startup.

The plugin library file base name is authentication_pam. The file name suffix differs per platform (for example, .so for Unix and Unix-like systems, .dll for Windows).

To load the plugin at server startup, use the --plugin-load-add option to name the library file that contains it. With this plugin-loading method, the option must be given each time the server starts. For example, put these lines in the server my.cnf file (adjust the .so suffix for your platform as necessary):

```
[mysqld]
plugin-load-add=authentication_pam.so
```

After modifying my.cnf, restart the server to cause the new settings to take effect.

Alternatively, to load the plugin at runtime, use this statement (adjust the .so suffix for your platform as necessary):

```
INSTALL PLUGIN authentication_pam SONAME 'authentication_pam.so';
```

INSTALL PLUGIN loads the plugin immediately, and also registers it in the mysql.plugins system table to cause the server to load it for each subsequent normal startup without the need for --plugin-load-add.

To verify plugin installation, examine the INFORMATION_SCHEMA.PLUGINS table or use the SHOW PLUGINS statement (see Obtaining Server Plugin Information). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
FROM INFORMATION_SCHEMA.PLUGINS
WHERE PLUGIN_NAME LIKE '%pam%';
```
Uninstalling PAM Pluggable Authentication

The method used to uninstall the PAM authentication plugin depends on how you installed it:

- If you installed the plugin at server startup using a `--plugin-load-add` option, restart the server without the option.
- If you installed the plugin at runtime using an `INSTALL PLUGIN` statement, it remains installed across server restarts. To uninstall it, use `UNINSTALL PLUGIN`:

  ```
  UNINSTALL PLUGIN authentication_pam;
  ```

Using PAM Pluggable Authentication

This section describes in general terms how to use the PAM authentication plugin to connect from MySQL client programs to the server. The following sections provide instructions for using PAM authentication in specific ways. It is assumed that the server is running with the server-side PAM plugin enabled, as described in Installing PAM Pluggable Authentication.

To refer to the PAM authentication plugin in the `IDENTIFIED WITH` clause of a `CREATE USER` statement, use the name `authentication_pam`. For example:

```sql
CREATE USER user
IDENTIFIED WITH authentication_pam
AS 'auth_string';
```

The authentication string specifies the following types of information:

- The PAM service name (see How PAM Authentication of MySQL Users Works). Examples in the following discussion use a service name of `mysql-unix` for authentication using traditional Unix passwords, and `mysql-ldap` for authentication using LDAP.
- For proxy support, PAM provides a way for a PAM module to return to the server a MySQL user name other than the external user name passed by the client program when it connects to the server. Use the authentication string to control the mapping from external user names to MySQL user names. If you want to take advantage of proxy user capabilities, the authentication string must include this kind of mapping.

For example, if an account uses the `mysql-unix` PAM service name and should map operating system users in the `root` and `users` PAM groups to the `developer` and `data_entry` MySQL users, respectively, use a statement like this:

```sql
CREATE USER user
IDENTIFIED WITH authentication_pam
AS 'mysql-unix, root=developer, users=data_entry';
```

Authentication string syntax for the PAM authentication plugin follows these rules:

- The string consists of a PAM service name, optionally followed by a PAM group mapping list consisting of one or more keyword/value pairs each specifying a PAM group name and a MySQL user name:

  ```
  pam_service_name[,pam_group_name=mysql_user_name]...
  ```

  The plugin parses the authentication string for each connection attempt that uses the account. To minimize overhead, keep the string as short as possible.
• Each `pam_group_name=mysql_user_name` pair must be preceded by a comma.

• Leading and trailing spaces not inside double quotation marks are ignored.

• Unquoted `pam_service_name`, `pam_group_name`, and `mysql_user_name` values can contain anything except equal sign, comma, or space.

• If a `pam_service_name`, `pam_group_name`, or `mysql_user_name` value is quoted with double quotation marks, everything between the quotation marks is part of the value. This is necessary, for example, if the value contains space characters. All characters are legal except double quotation mark and backslash (`\`). To include either character, escape it with a backslash.

If the plugin successfully authenticates the external user name (the name passed by the client), it looks for a PAM group mapping list in the authentication string and, if present, uses it to return a different MySQL user name to the MySQL server based on which PAM groups the external user is a member of:

• If the authentication string contains no PAM group mapping list, the plugin returns the external name.

• If the authentication string does contain a PAM group mapping list, the plugin examines each `pam_group_name=mysql_user_name` pair in the list from left to right and tries to find a match for the `pam_group_name` value in a non-MySQL directory of the groups assigned to the authenticated user and returns `mysql_user_name` for the first match it finds. If the plugin finds no match for any PAM group, it returns the external name. If the plugin is not capable of looking up a group in a directory, it ignores the PAM group mapping list and returns the external name.

The following sections describe how to set up several authentication scenarios that use the PAM authentication plugin:

• No proxy users. This uses PAM only to check login names and passwords. Every external user permitted to connect to MySQL Server should have a matching MySQL account that is defined to use PAM authentication. (For a MySQL account of `user_name@host_name` to match the external user, `user_name` must be the external user name and `host_name` must match the host from which the client connects.) Authentication can be performed by various PAM-supported methods. Later discussion shows how to authenticate client credentials using traditional Unix passwords, and passwords in LDAP.

PAM authentication, when not done through proxy users or PAM groups, requires the MySQL user name to be same as the operating system user name. MySQL user names are limited to 32 characters (see Section 4.3, “Grant Tables”), which limits PAM nonproxy authentication to Unix accounts with names of at most 32 characters.

• Proxy users only, with PAM group mapping. For this scenario, create one or more MySQL accounts that define different sets of privileges. (Ideally, nobody should connect using those accounts directly.) Then define a default user authenticating through PAM that uses some mapping scheme (usually based on the external PAM groups the users are members of) to map all the external user names to the few MySQL accounts holding the privilege sets. Any client who connects and specifies an external user name as the client user name is mapped to one of the MySQL accounts and uses its privileges. The discussion shows how to set this up using traditional Unix passwords, but other PAM methods such as LDAP could be used instead.

Variations on these scenarios are possible:

• You can permit some users to log in directly (without proxying) but require others to connect through proxy accounts.

• You can use one PAM authentication method for some users, and another method for other users, by using differing PAM service names among your PAM-authenticated accounts. For example, you can use the `mysql-unix` PAM service for some users, and `mysql-ldap` for others.

The examples make the following assumptions. You might need to make some adjustments if your system is set up differently.
The login name and password are *antonio* and *antonio_password*, respectively. Change these to correspond to the user you want to authenticate.

The PAM configuration directory is `/etc/pam.d`.

The PAM service name corresponds to the authentication method (mysql-unix or mysql-ldap in this discussion). To use a given PAM service, you must set up a PAM file with the same name in the PAM configuration directory (creating the file if it does not exist). In addition, you must name the PAM service in the authentication string of the `CREATE USER` statement for any account that authenticates using that PAM service.

The PAM authentication plugin checks at initialization time whether the `AUTHENTICATION_PAM_LOG` environment value is set in the server’s startup environment. If so, the plugin enables logging of diagnostic messages to the standard output. Depending on how your server is started, the message might appear on the console or in the error log. These messages can be helpful for debugging PAM-related issues that occur when the plugin performs authentication. For more information, see PAM Authentication Debugging.

### PAM Unix Password Authentication without Proxy Users

This authentication scenario uses PAM to check external users defined in terms of operating system user names and Unix passwords, without proxying. Every such external user permitted to connect to MySQL Server should have a matching MySQL account that is defined to use PAM authentication through traditional Unix password store.

Note

Traditional Unix passwords are checked using the `/etc/shadow` file. For information regarding possible issues related to this file, see PAM Authentication Access to Unix Password Store.

1. Verify that Unix authentication permits logins to the operating system with the user name *antonio* and password *antonio_password*.

2. Set up PAM to authenticate MySQL connections using traditional Unix passwords by creating a `mysql-unix` PAM service file named `/etc/pam.d/mysql-unix`. The file contents are system dependent, so check existing login-related files in the `/etc/pam.d` directory to see what they look like. On Linux, the `mysql-unix` file might look like this:

   ```
   # PAM-1.0
   auth            include         password-auth
   account         include         password-auth
   ```

   For macOS, use `login` rather than `password-auth`.

   The PAM file format might differ on some systems. For example, on Ubuntu and other Debian-based systems, use these file contents instead:

   ```
   @include common-auth
   @include common-account
   @include common-session-noninteractive
   ```

3. Create a MySQL account with the same user name as the operating system user name and define it to authenticate using the PAM plugin and the `mysql-unix` PAM service:

   ```
   CREATE USER 'antonio'@'localhost'
     IDENTIFIED WITH authentication_pam
     AS 'mysql-unix';
   GRANT ALL PRIVILEGES
    ON mydb.*
   TO 'antonio'@'localhost';
   ```

   Here, the authentication string contains only the PAM service name, `mysql-unix`, which authenticates Unix passwords.
4. Use the `mysql` command-line client to connect to the MySQL server as `antonio`. For example:

```
shell> mysql --user=antonio --password --enable-cleartext-plugin
Enter password: antonio_password
```

The server should permit the connection and the following query returns output as shown:

```
mysql> SELECT USER(), CURRENT_USER(), @@proxy_user;

<table>
<thead>
<tr>
<th>USER()</th>
<th>CURRENT_USER()</th>
<th>@@proxy_user</th>
</tr>
</thead>
<tbody>
<tr>
<td>antonio@localhost</td>
<td>antonio@localhost</td>
<td>NULL</td>
</tr>
</tbody>
</table>
```

This demonstrates that the `antonio` operating system user is authenticated to have the privileges granted to the `antonio` MySQL user, and that no proxying has occurred.

**Note**

The client-side `mysql_clear_password` authentication plugin leaves the password untouched, so client programs send it to the MySQL server as cleartext. This enables the password to be passed as is to PAM. A cleartext password is necessary to use the server-side PAM library, but may be a security problem in some configurations. These measures minimize the risk:

- To make inadvertent use of the `mysql_clear_password` plugin less likely, MySQL clients must explicitly enable it (for example, with the `--enable-cleartext-plugin` option). See Section 6.1.4, “Client-Side Cleartext Pluggable Authentication”.
- To avoid password exposure with the `mysql_clear_password` plugin enabled, MySQL clients should connect to the MySQL server using an encrypted connection. See Section 5.1, “Configuring MySQL to Use Encrypted Connections”.

### PAM LDAP Authentication without Proxy Users

This authentication scenario uses PAM to check external users defined in terms of operating system user names and LDAP passwords, without proxying. Every such external user permitted to connect to MySQL Server should have a matching MySQL account that is defined to use PAM authentication through LDAP.

To use PAM LDAP pluggable authentication for MySQL, these prerequisites must be satisfied:

- An LDAP server must be available for the PAM LDAP service to communicate with.
- LDAP users to be authenticated by MySQL must be present in the directory managed by the LDAP server.

**Note**

Another way to use LDAP for MySQL user authentication is to use the LDAP-specific authentication plugins. See Section 6.1.7, “LDAP Pluggable Authentication”.

Configure MySQL for PAM LDAP authentication as follows:

1. Verify that Unix authentication permits logins to the operating system with the user name `antonio` and password `antonio_password`.
2. Set up PAM to authenticate MySQL connections using LDAP by creating a `mysql-ldap` PAM service file named `/etc/pam.d/mysql-ldap`. The file contents are system dependent, so check existing login-related files in the `/etc/pam.d` directory to see what they look like. On Linux, the `mysql-ldap` file might look like this:
PAM Pluggable Authentication

```bash
#%PAM-1.0
auth required pam_ldap.so
account required pam_ldap.so
```

If PAM object files have a suffix different from .so on your system, substitute the correct suffix.

The PAM file format might differ on some systems.

3. Create a MySQL account with the same user name as the operating system user name and define it to authenticate using the PAM plugin and the mysql-ldap PAM service:

```sql
CREATE USER 'antonio'@'localhost'
    IDENTIFIED WITH authentication_pam
    AS 'mysql-ldap';
GRANT ALL PRIVILEGES
    ON mydb.*
    TO 'antonio'@'localhost';
```

Here, the authentication string contains only the PAM service name, mysql-ldap, which authenticates using LDAP.

4. Connecting to the server is the same as described in PAM Unix Password Authentication without Proxy Users.

PAM Unix Password Authentication with Proxy Users and Group Mapping

The authentication scheme described here uses proxying and PAM group mapping to map connecting MySQL users who authenticate using PAM onto other MySQL accounts that define different sets of privileges. Users do not connect directly through the accounts that define the privileges. Instead, they connect through a default proxy account authenticated using PAM, such that all the external users are mapped to the MySQL accounts that hold the privileges. Any user who connects using the proxy account is mapped to one of those MySQL accounts, the privileges for which determine the database operations permitted to the external user.

The procedure shown here uses Unix password authentication. To use LDAP instead, see the early steps of PAM LDAP Authentication without Proxy Users.

---

**Note**

Traditional Unix passwords are checked using the /etc/shadow file. For information regarding possible issues related to this file, see PAM Authentication Access to Unix Password Store.

1. Verify that Unix authentication permits logins to the operating system with the user name antonio and password antonio_password.

2. Verify that antonio is a member of the root or users PAM group.

3. Set up PAM to authenticate the mysql-unix PAM service through operating system users by creating a file named /etc/pam.d/mysql-unix. The file contents are system dependent, so check existing login-related files in the /etc/pam.d directory to see what they look like. On Linux, the mysql-unix file might look like this:

```bash
#%PAM-1.0
auth include password-auth
account include password-auth
```

For macOS, use login rather than password-auth.

The PAM file format might differ on some systems. For example, on Ubuntu and other Debian-based systems, use these file contents instead:

```bash
@include common-auth
```
4. Create a default proxy user (''@'') that maps external PAM users to the proxied accounts:

```sql
CREATE USER ''@''
    IDENTIFIED WITH authentication_pam
    AS 'mysql-unix, root=developer, users=data_entry';
```

Here, the authentication string contains the PAM service name, `mysql-unix`, which authenticates Unix passwords. The authentication string also maps external users in the `root` and `users` PAM groups to the `developer` and `data_entry` MySQL user names, respectively.

The PAM group mapping list following the PAM service name is required when you set up proxy users. Otherwise, the plugin cannot tell how to perform mapping from external user names to the proper proxied MySQL user names.

**Note**
If your MySQL installation has anonymous users, they might conflict with the default proxy user. For more information about this issue, and ways of dealing with it, see Default Proxy User and Anonymous User Conflicts.

5. Create the proxied accounts and grant to each one the privileges it should have:

```sql
CREATE USER 'developer'@'localhost'
    IDENTIFIED WITH mysql_no_login;
CREATE USER 'data_entry'@'localhost'
    IDENTIFIED WITH mysql_no_login;
GRANT ALL PRIVILEGES
    ON mydevdb.*
    TO 'developer'@'localhost';
GRANT ALL PRIVILEGES
    ON mydb.*
    TO 'data_entry'@'localhost';
```

The proxied accounts use the `mysql_no_login` authentication plugin to prevent clients from using the accounts to log in directly to the MySQL server. Instead, it is expected that users who authenticate using PAM will use the `developer` or `data_entry` account by proxy based on their PAM group. (This assumes that the plugin is installed. For instructions, see Section 6.1.8, "No-Login Pluggable Authentication".) For alternative methods of protecting proxied accounts against direct use, see Preventing Direct Login to Proxied Accounts.

6. Grant to the proxy account the `PROXY` privilege for each proxied account:

```sql
GRANT PROXY
    ON 'developer'@'localhost'
    TO ''@'';
GRANT PROXY
    ON 'data_entry'@'localhost'
    TO ''@'';
```

7. Use the `mysql` command-line client to connect to the MySQL server as `antonio`.

```
shell> mysql --user=antonio --password --enable-cleartext-plugin
Enter password: antonio_password
```

The server authenticates the connection using the default ''@' proxy account. The resulting privileges for `antonio` depend on which PAM groups `antonio` is a member of. If `antonio` is a member of the `root` PAM group, the PAM plugin maps `root` to the `developer` MySQL user name and returns that name to the server. The server verifies that ''@' has the PROXY privilege for `developer` and permits the connection. The following query returns output as shown:

```sql
mysql> SELECT USER(), CURRENT_USER(), @@proxy_user;
+-------------------+---------------------+--------------+
| USER()            | CURRENT_USER()      | @@proxy_user |
+-------------------+---------------------+--------------+
```

<table>
<thead>
<tr>
<th>USER()</th>
<th>CURRENT_USER()</th>
<th>@@proxy_user</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
This demonstrates that the antonio operating system user is authenticated to have the privileges granted to the developer MySQL user, and that proxying occurs through the default proxy account.

If antonio is not a member of the root PAM group but is a member of the users PAM group, a similar process occurs, but the plugin maps user PAM group membership to the data_entry MySQL user name and returns that name to the server:

```
mysql> SELECT USER(), CURRENT_USER(), @@proxy_user;
+-------------------+----------------------+--------------+
| USER()            | CURRENT_USER()       | @@proxy_user |
|-------------------|----------------------|--------------+
| antonio@localhost | data_entry@localhost | ''@''        |
+-------------------+----------------------|--------------+
```

This demonstrates that the antonio operating system user is authenticated to have the privileges of the data_entry MySQL user, and that proxying occurs through the default proxy account.

Note

The client-side mysql_clear_password authentication plugin leaves the password untouched, so client programs send it to the MySQL server as cleartext. This enables the password to be passed as is to PAM. A cleartext password is necessary to use the server-side PAM library, but may be a security problem in some configurations. These measures minimize the risk:

- To make inadvertent use of the mysql_clear_password plugin less likely, MySQL clients must explicitly enable it (for example, with the --enable-cleartext-plugin option). See Section 6.1.4, “Client-Side Cleartext Pluggable Authentication”.
- To avoid password exposure with the mysql_clear_password plugin enabled, MySQL clients should connect to the MySQL server using an encrypted connection. See Section 5.1, “Configuring MySQL to Use Encrypted Connections”.

PAM Authentication Access to Unix Password Store

On some systems, Unix authentication uses a password store such as /etc/shadow, a file that typically has restricted access permissions. This can cause MySQL PAM-based authentication to fail. Unfortunately, the PAM implementation does not permit distinguishing “password could not be checked” (due, for example, to inability to read /etc/shadow) from “password does not match.” If you are using Unix password store for PAM authentication, you may be able to enable access to it from MySQL using one of the following methods:

- Assuming that the MySQL server is run from the mysql operating system account, put that account in the shadow group that has /etc/shadow access:

  1. Create a shadow group in /etc/group.
  2. Add the mysql operating system user to the shadow group in /etc/group.
  3. Assign /etc/group to the shadow group and enable the group read permission:

```
chgrp shadow /etc/shadow
chmod g+r /etc/shadow
```
  4. Restart the MySQL server.
• If you are using the `pam_unix` module and the `unix_chkpwd` utility, enable password store access as follows:

```bash
chmod u-s /usr/sbin/unix_chkpwd
setcap cap_dac_read_search+ep /usr/sbin/unix_chkpwd
```

Adjust the path to `unix_chkpwd` as necessary for your platform.

## PAM Authentication Debugging

The PAM authentication plugin checks at initialization time whether the `AUTHENTICATION_PAM_LOG` environment value is set (the value does not matter). If so, the plugin enables logging of diagnostic messages to the standard output. These messages may be helpful for debugging PAM-related issues that occur when the plugin performs authentication.

Some messages include reference to PAM plugin source files and line numbers, which enables plugin actions to be tied more closely to the location in the code where they occur.

Another technique for debugging connection failures and determining what is happening during connection attempts is to configure PAM authentication to permit all connections, then check the system log files. This technique should be used only on a temporary basis, and not on a production server.

Configure a PAM service file named `/etc/pam.d/mysql-any-password` with these contents (the format may differ on some systems):

```bash
#%PAM-1.0
auth        required    pam_permit.so
account     required    pam_permit.so
```

Create an account that uses the PAM plugin and names the `mysql-any-password` PAM service:

```sql
CREATE USER 'testuser'@'localhost'
  IDENTIFIED WITH authentication_pam
  AS 'mysql-any-password';
```

The `mysql-any-password` service file causes any authentication attempt to return true, even for incorrect passwords. If an authentication attempt fails, that tells you the configuration problem is on the MySQL side. Otherwise, the problem is on the operating system/PAM side. To see what might be happening, check system log files such as `/var/log/secure`, `/var/log/audit.log`, `/var/log/syslog`, or `/var/log/messages`.

After determining what the problem is, remove the `mysql-any-password` PAM service file to disable any-password access.

### 6.1.6 Windows Pluggable Authentication

**Note**

Windows pluggable authentication is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see [https://www.mysql.com/products/](https://www.mysql.com/products/).

MySQL Enterprise Edition for Windows supports an authentication method that performs external authentication on Windows, enabling MySQL Server to use native Windows services to authenticate client connections. Users who have logged in to Windows can connect from MySQL client programs to the server based on the information in their environment without specifying an additional password.

The client and server exchange data packets in the authentication handshake. As a result of this exchange, the server creates a security context object that represents the identity of the client in the Windows OS. This identity includes the name of the client account. Windows pluggable authentication uses the identity of the client to check whether it is a given account or a member of a group. By default, negotiation uses Kerberos to authenticate, then NTLM if Kerberos is unavailable.
Windows Pluggable Authentication

Windows pluggable authentication provides these capabilities:

- External authentication: Windows authentication enables MySQL Server to accept connections from users defined outside the MySQL grant tables who have logged in to Windows.

- Proxy user support: Windows authentication can return to MySQL a user name different from the external user name passed by the client program. This means that the plugin can return the MySQL user that defines the privileges the external Windows-authenticated user should have. For example, a Windows user named joe can connect and have the privileges of a MySQL user named developer.

The following table shows the plugin and library file names. The file must be located in the directory named by the `plugin_dir` system variable.

Table 6.6 Plugin and Library Names for Windows Authentication

<table>
<thead>
<tr>
<th>Plugin or File</th>
<th>Plugin or File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server-side plugin</td>
<td><code>authentication_windows</code></td>
</tr>
<tr>
<td>Client-side plugin</td>
<td><code>authentication_windows_client</code></td>
</tr>
<tr>
<td>Library file</td>
<td><code>authentication_windows.dll</code></td>
</tr>
</tbody>
</table>

The library file includes only the server-side plugin. The client-side plugin is built into the `libmysqlclient` client library.

The server-side Windows authentication plugin is included only in MySQL Enterprise Edition. It is not included in MySQL community distributions. The client-side plugin is included in all distributions, including community distributions. This permits clients from any distribution to connect to a server that has the server-side plugin loaded.

The Windows authentication plugin is supported on any version of Windows supported by MySQL 8.0 (see https://www.mysql.com/support/supportedplatforms/database.html).

The following sections provide installation and usage information specific to Windows pluggable authentication:

- Installing Windows Pluggable Authentication
- Uninstalling Windows Pluggable Authentication
- Using Windows Pluggable Authentication

For general information about pluggable authentication in MySQL, see Section 4.17, “Pluggable Authentication”. For proxy user information, see Section 4.18, “Proxy Users”.

Installing Windows Pluggable Authentication

This section describes how to install the Windows authentication plugin. For general information about installing plugins, see Installing and Uninstalling Plugins.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

To load the plugin at server startup, use the `--plugin-load-add` option to name the library file that contains it. With this plugin-loading method, the option must be given each time the server starts. For example, put these lines in the server `my.cnf` file:

```
[mysqld]
plugin-load-add=authentication_windows.dll
```

After modifying `my.cnf`, restart the server to cause the new settings to take effect.

Alternatively, to load the plugin at runtime, use this statement:
INSTALL PLUGIN authentication_windows SONAME 'authentication_windows.dll';

INSTALL PLUGIN loads the plugin immediately, and also registers it in the mysql.plugins system table to cause the server to load it for each subsequent normal startup without the need for --plugin-load-add.

To verify plugin installation, examine the INFORMATION_SCHEMA.PLUGINS table or use the SHOW PLUGINS statement (see Obtaining Server Plugin Information). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
FROM INFORMATION_SCHEMA.PLUGINS
WHERE PLUGIN_NAME LIKE '%windows%';
```

+------------------------+---------------+
<table>
<thead>
<tr>
<th>PLUGIN_NAME</th>
<th>PLUGIN_STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>authentication_windows</td>
<td>ACTIVE</td>
</tr>
</tbody>
</table>

If the plugin fails to initialize, check the server error log for diagnostic messages.

To associate MySQL accounts with the Windows authentication plugin, see Using Windows Pluggable Authentication. Additional plugin control is provided by the authentication_windows_use_principal_name and authentication_windows_log_level system variables. See Server System Variables.

### Uninstalling Windows Pluggable Authentication

The method used to uninstall the Windows authentication plugin depends on how you installed it:

- If you installed the plugin at server startup using a --plugin-load-add option, restart the server without the option.

- If you installed the plugin at runtime using an INSTALL PLUGIN statement, it remains installed across server restarts. To uninstall it, use UNINSTALL PLUGIN:

```
UNINSTALL PLUGIN authentication_windows;
```

In addition, remove any startup options that set Windows plugin-related system variables.

### Using Windows Pluggable Authentication

The Windows authentication plugin supports the use of MySQL accounts such that users who have logged in to Windows can connect to the MySQL server without having to specify an additional password. It is assumed that the server is running with the server-side plugin enabled, as described in Installing Windows Pluggable Authentication. Once the DBA has enabled the server-side plugin and set up accounts to use it, clients can connect using those accounts with no other setup required on their part.

To refer to the Windows authentication plugin in the IDENTIFIED WITH clause of a CREATE USER statement, use the name authentication_windows. Suppose that the Windows users Rafal and Tasha should be permitted to connect to MySQL, as well as any users in the Administrators or Power Users group. To set this up, create a MySQL account named sql_admin that uses the Windows plugin for authentication:

```
CREATE USER sql_admin
IDENTIFIED WITH authentication_windows
AS 'Rafal, Tasha, Administrators, "Power Users"';
```

The plugin name is authentication_windows. The string following the AS keyword is the authentication string. It specifies that the Windows users named Rafal or Tasha are permitted to authenticate to the server as the MySQL user sql_admin, as are any Windows users in the Administrators or Power Users group. The latter group name contains a space, so it must be quoted with double quote characters.
After you create the `sql_admin` account, a user who has logged in to Windows can attempt to connect to the server using that account:

```
C:\> mysql --user=sql_admin
```

No password is required here. The `authentication_windows` plugin uses the Windows security API to check which Windows user is connecting. If that user is named Rafal or Tasha, or is a member of the `Administrators` or `Power Users` group, the server grants access and the client is authenticated as `sql_admin` and has whatever privileges are granted to the `sql_admin` account. Otherwise, the server denies access.

Authentication string syntax for the Windows authentication plugin follows these rules:

- The string consists of one or more user mappings separated by commas.
- Each user mapping associates a Windows user or group name with a MySQL user name:
  ```
  win_user_or_group_name=mysql_user_name
  win_user_or_group_name
  ```
  For the latter syntax, with no `mysql_user_name` value given, the implicit value is the MySQL user created by the `CREATE USER` statement. Thus, these statements are equivalent:

```
CREATE USER sql_admin
    IDENTIFIED WITH authentication_windows
    AS 'Rafal, Tasha, Administrators, "Power Users"';
CREATE USER sql_admin
    IDENTIFIED WITH authentication_windows
    AS 'Rafal=sql_admin, Tasha=sql_admin, Administrators=sql_admin, "Power Users"=sql_admin';
```

- Each backslash character (`\`) in a value must be doubled because backslash is the escape character in MySQL strings.
- Leading and trailing spaces not inside double quotation marks are ignored.
- Unquoted `win_user_or_group_name` and `mysql_user_name` values can contain anything except equal sign, comma, or space.
- If a `win_user_or_group_name` and or `mysql_user_name` value is quoted with double quotation marks, everything between the quotation marks is part of the value. This is necessary, for example, if the name contains space characters. All characters within double quotes are legal except double quotation mark and backslash. To include either character, escape it with a backslash.
- `win_user_or_group_name` values use conventional syntax for Windows principals, either local or in a domain. Examples (note the doubling of backslashes):

```
domain\user
.\user
domain\group
.\group
BUILTIN\WellKnownGroup
```

When invoked by the server to authenticate a client, the plugin scans the authentication string left to right for a user or group match to the Windows user. If there is a match, the plugin returns the corresponding `mysql_user_name` to the MySQL server. If there is no match, authentication fails.

A user name match takes preference over a group name match. Suppose that the Windows user named `win_user` is a member of `win_group` and the authentication string looks like this:

```
'win_group = sql_user1, win_user = sql_user2'
```

When `win_user` connects to the MySQL server, there is a match both to `win_group` and to `win_user`. The plugin authenticates the user as `sql_user2` because the more-specific user match takes precedence over the group match, even though the group is listed first in the authentication string.
Windows authentication always works for connections from the same computer on which the server is running. For cross-computer connections, both computers must be registered with Windows Active Directory. If they are in the same Windows domain, it is unnecessary to specify a domain name. It is also possible to permit connections from a different domain, as in this example:

```
CREATE USER sql_accounting
  IDENTIFIED WITH authentication_windows
  AS 'SomeDomain\Accounting';
```

Here `SomeDomain` is the name of the other domain. The backslash character is doubled because it is the MySQL escape character within strings.

MySQL supports the concept of proxy users whereby a client can connect and authenticate to the MySQL server using one account but while connected has the privileges of another account (see Section 4.18, “Proxy Users”). Suppose that you want Windows users to connect using a single user name but be mapped based on their Windows user and group names onto specific MySQL accounts as follows:

- The `local_user` and `MyDomain\domain_user` local and domain Windows users should map to the `local_wlad` MySQL account.
- Users in the `MyDomain\Developers` domain group should map to the `local_dev` MySQL account.
- Local machine administrators should map to the `local_admin` MySQL account.

To set this up, create a proxy account for Windows users to connect to, and configure this account so that users and groups map to the appropriate MySQL accounts (`local_wlad`, `local_dev`, `local_admin`). In addition, grant the MySQL accounts the privileges appropriate to the operations they need to perform. The following instructions use `win_proxy` as the proxy account, and `local_wlad`, `local_dev`, and `local_admin` as the proxied accounts.

1. Create the proxy MySQL account:

```
CREATE USER win_proxy
  IDENTIFIED WITH authentication_windows
  AS 'local_user = local_wlad,
  MyDomain\domain_user = local_wlad,
  MyDomain\Developers = local_dev,
  BUILTIN\Administrators = local_admin';
```

2. For proxying to work, the proxied accounts must exist, so create them:

```
CREATE USER local_wlad
  IDENTIFIED WITH mysql_no_login;
CREATE USER local_dev
  IDENTIFIED WITH mysql_no_login;
CREATE USER local_admin
  IDENTIFIED WITH mysql_no_login;
```

The proxied accounts use the `mysql_no_login` authentication plugin to prevent clients from using the accounts to log in directly to the MySQL server. Instead, it is expected that users who authenticate using Windows will use the `win_proxy` proxy account. (This assumes that the plugin is installed. For instructions, see Section 6.1.8, “No-Login Pluggable Authentication”.) For alternative methods of protecting proxied accounts against direct use, see Preventing Direct Login to Proxied Accounts.

You should also execute `GRANT` statements (not shown) that grant each proxied account the privileges required for MySQL access.

3. Grant to the proxy account the `PROXY` privilege for each proxied account:

```
GRANT PROXY ON local_wlad TO win_proxy;
GRANT PROXY ON local_dev TO win_proxy;
GRANT PROXY ON local_admin TO win_proxy;
```
Now the Windows users local_user and MyDomain\domain_user can connect to the MySQL server as win_proxy and when authenticated have the privileges of the account given in the authentication string (in this case, local_wlad). A user in the MyDomain\Developers group who connects as win_proxy has the privileges of the local_dev account. A user in the BUILTIN \Administrators group has the privileges of the local_admin account.

To configure authentication so that all Windows users who do not have their own MySQL account go through a proxy account, substitute the default proxy account (''@'') for win_proxy in the preceding instructions. For information about default proxy accounts, see Section 4.18, “Proxy Users”.

Note
If your MySQL installation has anonymous users, they might conflict with the default proxy user. For more information about this issue, and ways of dealing with it, see Default Proxy User and Anonymous User Conflicts.

To use the Windows authentication plugin with Connector/NET connection strings in Connector/NET 6.4.4 and higher, see Using the Windows Native Authentication Plugin.

6.1.7 LDAP Pluggable Authentication

Note
LDAP pluggable authentication is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see https://www.mysql.com/products/.

MySQL Enterprise Edition supports an authentication method that enables MySQL Server to use LDAP (Lightweight Directory Access Protocol) to authenticate MySQL users by accessing directory services such as X.500. MySQL uses LDAP to fetch user, credential, and group information.

LDAP pluggable authentication provides these capabilities:

• External authentication: LDAP authentication enables MySQL Server to accept connections from users defined outside the MySQL grant tables in LDAP directories.

• Proxy user support: LDAP authentication can return to MySQL a user name different from the external user name passed by the client program, based on the LDAP groups the external user is a member of. This means that an LDAP plugin can return the MySQL user that defines the privileges the external LDAP-authenticated user should have. For example, an LDAP user named joe can connect and have the privileges of a MySQL user named developer, if the LDAP group for joe is developer.

• Security: Using TLS, connections to the LDAP server can be secure.

The following tables show the plugin and library file names for simple and SASL-based LDAP authentication. The file name suffix might differ on your system. The files must be located in the directory named by the plugin_dir system variable.

Table 6.7 Plugin and Library Names for Simple LDAP Authentication

<table>
<thead>
<tr>
<th>Plugin or File</th>
<th>Plugin or File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server-side plugin name</td>
<td>authentication_ldap_simple</td>
</tr>
<tr>
<td>Client-side plugin name</td>
<td>mysql_clear_password</td>
</tr>
<tr>
<td>Library file name</td>
<td>authentication_ldap_simple.so</td>
</tr>
</tbody>
</table>

Table 6.8 Plugin and Library Names for SASL-Based LDAP Authentication

<table>
<thead>
<tr>
<th>Plugin or File</th>
<th>Plugin or File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server-side plugin name</td>
<td>authentication_ldap_sasl</td>
</tr>
</tbody>
</table>
### LDAP Pluggable Authentication

The library files include only the `authentication_ldap_XXX` authentication plugins. The client-side `mysql_clear_password` plugin is built into the `libmysqlclient` client library.

Each server-side LDAP plugin works with a specific client-side plugin:

- **The server-side `authentication_ldap_simple` plugin** performs simple LDAP authentication. For connections by accounts that use this plugin, client programs use the client-side `mysql_clear_password` plugin, which sends the password to the server as cleartext. No password hashing or encryption is used, so a secure connection between the MySQL client and server is recommended to prevent password exposure.

- **The server-side `authentication_ldap_sasl` plugin** performs SASL-based LDAP authentication. For connections by accounts that use this plugin, client programs use the client-side `authentication_ldap_sasl_client` plugin. The client-side and server-side SASL LDAP plugins use SASL messages for secure transmission of credentials within the LDAP protocol, to avoid sending the cleartext password between the MySQL client and server.

The following sections provide installation and usage information specific to LDAP pluggable authentication:

- Prerequisites for LDAP Pluggable Authentication
- How LDAP Authentication of MySQL Users Works
- Installing LDAP Pluggable Authentication
- Uninstalling LDAP Pluggable Authentication
- Using LDAP Pluggable Authentication
- Simple LDAP Authentication
- SASL-Based LDAP Authentication
- LDAP Authentication with Proxying
- LDAP Authentication Group Preference and Mapping Specification
- LDAP Authentication User DN Suffixes
- LDAP Authentication Methods
- The GSSAPI/Kerberos Authentication Method
- LDAP Search Referral

For general information about pluggable authentication in MySQL, see Section 4.17, “Pluggable Authentication”. For information about the `mysql_clear_password` plugin, see Section 6.1.4, “Client-Side Cleartext Pluggable Authentication”. For proxy user information, see Section 4.18, “Proxy Users”.

---

**Note**

If your system supports PAM and permits LDAP as a PAM authentication method, another way to use LDAP for MySQL user authentication is to use the server-side `authentication_pam` plugin. See Section 6.1.5, “PAM Pluggable Authentication”.

---

### Plugin or File

<table>
<thead>
<tr>
<th>Plugin or File Name</th>
<th>Plugin or File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client-side plugin name</td>
<td><code>authentication_ldap_sasl_client</code></td>
</tr>
<tr>
<td>Library file names</td>
<td><code>authentication_ldap_sasl.so</code>, <code>authentication_ldap_sasl_client.so</code></td>
</tr>
</tbody>
</table>
Prerequisites for LDAP Pluggable Authentication

To use LDAP pluggable authentication for MySQL, these prerequisites must be satisfied:

- An LDAP server must be available for the LDAP authentication plugins to communicate with.
- LDAP users to be authenticated by MySQL must be present in the directory managed by the LDAP server.
- An LDAP client library must be available on systems where the server-side `authentication_ldap_sasl` or `authentication_ldap_simple` plugin is used. Currently, supported libraries are the Windows native LDAP library, or the OpenLDAP library on non-Windows systems.
- To use SASL-based LDAP authentication:
  - The LDAP server must be configured to communicate with a SASL server.
  - A SASL client library must be available on systems where the client-side `authentication_ldap_sasl_client` plugin is used. Currently, the only supported library is the Cyrus SASL library.
- To use a particular SASL authentication method, any other services required by that method must be available. For example, to use GSSAPI/Kerberos, a GSSAPI library and Kerberos services must be available.

How LDAP Authentication of MySQL Users Works

This section provides a general overview of how MySQL and LDAP work together to authenticate MySQL users. For examples showing how to set up MySQL accounts to use specific LDAP authentication plugins, see Using LDAP Pluggable Authentication. For information about authentication methods available to the LDAP plugins, see LDAP Authentication Methods.

The client connects to the MySQL server, providing the MySQL client user name and a password:

- For simple LDAP authentication, the client-side and server-side plugins communicate the password as cleartext. A secure connection between the MySQL client and server is recommended to prevent password exposure.
- For SASL-based LDAP authentication, the client-side and server-side plugins avoid sending the cleartext password between the MySQL client and server. For example, the plugins might use SASL messages for secure transmission of credentials within the LDAP protocol. For the GSSAPI authentication method, the client-side and server-side plugins communicate securely using Kerberos without using LDAP messages directly.

If the client user name and host name match no MySQL account, the connection is rejected.

If there is a matching MySQL account, authentication against LDAP occurs. The LDAP server looks for an entry matching the user and authenticates the entry against the LDAP password:

- If the MySQL account names an LDAP user distinguished name (DN), LDAP authentication uses that value and the LDAP password provided by the client. (To associate an LDAP user DN with a MySQL account, include a `BY` clause that specifies an authentication string in the `CREATE USER` statement that creates the account.)
- If the MySQL account names no LDAP user DN, LDAP authentication uses the user name and LDAP password provided by the client. In this case, the authentication plugin first binds to the LDAP server using the root DN and password as credentials to find the user DN based on the client user name, then authenticates that user DN against the LDAP password. This bind using the root credentials fails if the root DN and password are set to incorrect values, or are empty (not set) and the LDAP server does not permit anonymous connections.
LDAP Pluggable Authentication

If the LDAP server finds no match or multiple matches, authentication fails and the client connection is rejected.

If the LDAP server finds a single match, LDAP authentication succeeds (assuming that the password is correct), the LDAP server returns the LDAP entry, and the authentication plugin determines the name of the authenticated user based on that entry:

- If the LDAP entry has a group attribute (by default, the `cn` attribute), the plugin returns its value as the authenticated user name.
- If the LDAP entry has no group attribute, the authentication plugin returns the client user name as the authenticated user name.

The MySQL server compares the client user name with the authenticated user name to determine whether proxying occurs for the client session:

- If the names are the same, no proxying occurs: The MySQL account matching the client user name is used for privilege checking.
- If the names differ, proxying occurs: MySQL looks for an account matching the authenticated user name. That account becomes the proxied user, which is used for privilege checking. The MySQL account that matched the client user name is treated as the external proxy user.

## Installing LDAP Pluggable Authentication

This section describes how to install the LDAP authentication plugins. For general information about installing plugins, see [Installing and Uninstalling Plugins](#).

To be usable by the server, the plugin library files must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

The server-side plugin library file base names are `authentication_ldap_simple` and `authentication_ldap_sasl`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

To load the plugins at server startup, use `--plugin-load-add` options to name the library files that contain them. With this plugin-loading method, the options must be given each time the server starts. Also, specify values for any plugin-provided system variables you wish to configure.

Each server-side LDAP plugin exposes a set of system variables that enable its operation to be configured. Setting most of these is optional, but you must set the variables that specify the LDAP server host (so the plugin knows where to connect) and base distinguished name for LDAP bind operations (to limit the scope of searches and obtain faster searches). For details about all LDAP system variables, see [Section 6.1.11, “Pluggable Authentication System Variables”](#).

To load the plugins and set the LDAP server host and base distinguished name for LDAP bind operations, put lines such as these in your `my.cnf` file (adjust the `.so` suffix for your platform as necessary):

```sql
[mysqld]
plugin-load-add=authentication_ldap_simple.so
authentication_ldap_simple_server_host=127.0.0.1
authentication_ldap_simple_bind_base_dn="dc=example,dc=com"
plugin-load-add=authentication_ldap_sasl.so
authentication_ldap_sasl_server_host=127.0.0.1
authentication_ldap_sasl_bind_base_dn="dc=example,dc=com"
```

After modifying `my.cnf`, restart the server to cause the new settings to take effect.

Alternatively, to load the plugins at runtime, use these statements (adjust the `.so` suffix for your platform as necessary):

```sql
INSTALL PLUGIN authentication_ldap_simple
```

171
LDAP Pluggable Authentication

SONAME 'authentication_ldap_simple.so';
INSTALL PLUGIN authentication_ldap_sasl
SONAME 'authentication_ldap_sasl.so';

`INSTALL PLUGIN` loads the plugin immediately, and also registers it in the `mysql.plugins` system table to cause the server to load it for each subsequent normal startup without the need for `--plugin-load-add`.

After installing the plugins at runtime, their system variables become available and you can add settings for them to your `my.cnf` file to configure the plugins for subsequent restarts. For example:

```
[mysqld]
authentication_ldap_simple_server_host=127.0.0.1
authentication_ldap_simple_bind_base_dn="dc=example,dc=com"
authentication_ldap_sasl_server_host=127.0.0.1
authentication_ldap_sasl_bind_base_dn="dc=example,dc=com"
```

After modifying `my.cnf`, restart the server to cause the new settings to take effect.

Alternatively, to set and persist the values at runtime, use these statements:

```sql
SET PERSIST authentication_ldap_simple_server_host='127.0.0.1';
SET PERSIST authentication_ldap_simple_bind_base_dn='dc=example,dc=com';
SET PERSIST authentication_ldap_sasl_server_host='127.0.0.1';
SET PERSIST authentication_ldap_sasl_bind_base_dn='dc=example,dc=com';
```

`SET PERSIST` sets the value for the running MySQL instance. It also saves the value, causing it to be used for subsequent server restarts. To change a value for the running MySQL instance without saving it for subsequent restarts, use the `GLOBAL` keyword rather than `PERSIST`. See `SET Syntax for Variable Assignment`.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement (see Obtaining Server Plugin Information). For example:

```sql
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
FROM INFORMATION_SCHEMA.PLUGINS
WHERE PLUGIN_NAME LIKE '%ldap%';
```

<table>
<thead>
<tr>
<th>PLUGIN_NAME</th>
<th>PLUGIN_STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>authentication_ldap_sasl</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>authentication_ldap_simple</td>
<td>ACTIVE</td>
</tr>
</tbody>
</table>

If a plugin fails to initialize, check the server error log for diagnostic messages.

To associate MySQL accounts with an LDAP plugin, see Using LDAP Pluggable Authentication.

---

**Additional Notes for SELinux**

On systems running EL6 or EL that have SELinux enabled, changes to the SELinux policy are required to enable the MySQL LDAP plugins to communicate with the LDAP service:

1. Create a file `mysqlldap.te` with these contents:

   ```
   module mysqlldap 1.0;
   require {
   type ldap_port_t;
   type mysqld_t;
   class tcp_socket name_connect;
   }
   #=============== mysqld_t ================
   allow mysqld_t ldap_port_t:tcp_socket name_connect;
   ```

2. Compile the security policy module into a binary representation:

   ```bash
   checkmodule -M -m mysqlldap.te -o mysqlldap.mod
   ```
3. Create an SELinux policy module package:

   ```bash
   semodule_package -m mysqlldap.mod -o mysqlldap.pp
   ```

4. Install the module package:

   ```bash
   semodule -i mysqlldap.pp
   ```

5. When the SELinux policy changes have been made, restart the MySQL server:

   ```bash
   service mysqld restart
   ```

### Uninstalling LDAP Pluggable Authentication

The method used to uninstall the LDAP authentication plugins depends on how you installed them:

- If you installed the plugins at server startup using `--plugin-load-add` options, restart the server without those options.

- If you installed the plugins at runtime using `INSTALL PLUGIN`, they remain installed across server restarts. To uninstall them, use `UNINSTALL PLUGIN`:

  ```
  UNINSTALL PLUGIN authentication_ldap_simple;
  UNINSTALL PLUGIN authentication_ldap_sasl;
  ```

In addition, remove from your `my.cnf` file any startup options that set LDAP plugin-related system variables. If you used `SET PERSIST` to persist LDAP system variables, use `RESET PERSIST` to remove the settings.

### Using LDAP Pluggable Authentication

This section describes how to enable MySQL accounts to connect to the MySQL server using LDAP pluggable authentication. It is assumed that the server is running with the appropriate server-side plugins enabled, as described in Installing LDAP Pluggable Authentication, and that the appropriate client-side plugins are available on the client host.

This section does not describe LDAP configuration or administration. You are assumed to be familiar with those topics.

The two server-side LDAP plugins each work with a specific client-side plugin:

- The server-side `authentication_ldap_simple` plugin performs simple LDAP authentication. For connections by accounts that use this plugin, client programs use the client-side `mysql_clear_password` plugin, which sends the password to the server as cleartext. No password hashing or encryption is used, so a secure connection between the MySQL client and server is recommended to prevent password exposure.

- The server-side `authentication_ldap_sasl` plugin performs SASL-based LDAP authentication. For connections by accounts that use this plugin, client programs use the client-side `authentication_ldap_sasl_client` plugin. The client-side and server-side SASL LDAP plugins use SASL messages for secure transmission of credentials within the LDAP protocol, to avoid sending the cleartext password between the MySQL client and server.

Overall requirements for LDAP authentication of MySQL users:

- There must be an LDAP directory entry for each user to be authenticated.

- There must be a MySQL user account that specifies a server-side LDAP authentication plugin and optionally names the associated LDAP user distinguished name (DN). (To associate an LDAP user DN with a MySQL account, include a `BY` clause in the `CREATE USER` statement that creates the account.) If an account names no LDAP string, LDAP authentication uses the user name specified by the client to find the LDAP entry.
• Client programs connect using the connection method appropriate for the server-side authentication plugin the MySQL account uses. For LDAP authentication, connections require the MySQL user name and LDAP password. In addition, for accounts that use the server-side authentication_ldap_simple plugin, invoke client programs with the --enable-cleartext-plugin option to enable the client-side mysql_clear_password plugin.

The instructions here assume the following scenario:

• MySQL users betsy and boris authenticate to the LDAP entries for betsy_ldap and boris_ldap, respectively. (It is not necessary that the MySQL and LDAP user names differ. The use of different names in this discussion helps clarify whether an operation context is MySQL or LDAP.)

• LDAP entries use the uid attribute to specify user names. This may vary depending on LDAP server. Some LDAP servers use the cn attribute for user names rather than uid. To change the attribute, modify the authentication_ldap_simple_user_search_attr or authentication_ldap_sasl_user_search_attr system variable appropriately.

• These LDAP entries are available in the directory managed by the LDAP server, to provide distinguished name values that uniquely identify each user:

  ```
  uid=betsy_ldap,ou=People,dc=example,dc=com
  uid=boris_ldap,ou=People,dc=example,dc=com
  ```

• CREATE USER statements that create MySQL accounts name an LDAP user in the BY clause, to indicate which LDAP entry the MySQL account authenticates against.

The instructions for setting up an account that uses LDAP authentication depend on which server-side LDAP plugin is used. The following sections describe several usage scenarios.

**Simple LDAP Authentication**

To configure a MySQL account for simple LDAP authentication, the CREATE USER statement specifies the authentication_ldap_simple plugin, and optionally names the LDAP user distinguished name (DN):

```
CREATE USER user
  IDENTIFIED WITH authentication_ldap_simple
  [BY 'LDAP user DN'];
```

Suppose that MySQL user betsy has this entry in the LDAP directory:

```
uid=betsy_ldap,ou=People,dc=example,dc=com
```

Then the statement to create the MySQL account for betsy looks like this:

```
CREATE USER 'betsy'@'localhost'
  IDENTIFIED WITH authentication_ldap_simple
  AS 'uid=betsy_ldap,ou=People,dc=example,dc=com';
```

The authentication string specified in the BY clause does not include the LDAP password. That must be provided by the client user at connect time.

Clients connect to the MySQL server by providing the MySQL user name and LDAP password, and by enabling the client-side mysql_clear_password plugin:

```
shell> mysql --user=betsy --password --enable-cleartext-plugin
Enter password: betsy_password (betsy_ldap LDAP password)
```

**Note**

The client-side mysql_clear_password authentication plugin leaves the password untouched, so client programs send it to the MySQL server as cleartext. This enables the password to be passed as is to the LDAP server. A cleartext password is necessary to use the server-side LDAP library without
LDAP Pluggable Authentication

SASL, but may be a security problem in some configurations. These measures minimize the risk:

- To make inadvertent use of the `mysql_clear_password` plugin less likely, MySQL clients must explicitly enable it (for example, with the `--enable-cleartext-plugin` option). See Section 6.1.4, “Client-Side Cleartext Pluggable Authentication”.

- To avoid password exposure with the `mysql_clear_password` plugin enabled, MySQL clients should connect to the MySQL server using an encrypted connection. See Section 5.1, “Configuring MySQL to Use Encrypted Connections”.

The authentication process occurs as follows:

1. The client-side plugin sends `betsy` and `betsy_password` as the client user name and LDAP password to the MySQL server.

2. The connection attempt matches the `betsy@localhost` account. The server-side LDAP plugin finds that this account has an authentication string of `uid=betsy_ldap,ou=People,dc=example,dc=com` to name the LDAP user DN. The plugin sends this string and the LDAP password to the LDAP server.

3. The LDAP server finds the LDAP entry for `betsy_ldap` and the password matches, so LDAP authentication succeeds.

4. The LDAP entry has no group attribute, so the server-side plugin returns the client user name (`betsy`) as the authenticated user. This is the same user name supplied by the client, so no proxying occurs and the client session uses the `betsy@localhost` account for privilege checking.

Had the matching LDAP entry contained a group attribute, that attribute value would have been the authenticated user name and, if the value differed from `betsy`, proxying would have occurred. For examples that use the group attribute, see LDAP Authentication with Proxying.

Had the `CREATE USER` statement contained no `BY` clause to specify the `betsy_ldap` LDAP distinguished name, authentication attempts would use the user name provided by the client (in this case, `betsy`). In the absence of an LDAP entry for `betsy`, authentication would fail.

SASL-Based LDAP Authentication

To configure a MySQL account for SASL LDAP authentication, the `CREATE USER` statement specifies the `authentication_ldap_sasl` plugin, and optionally names the LDAP user distinguished name (DN):

```
CREATE USER user
  IDENTIFIED WITH authentication_ldap_sasl
  [BY 'LDAP user DN'];
```

Suppose that MySQL user `boris` has this entry in the LDAP directory:

```
uid=boris_ldap,ou=People,dc=example,dc=com
```

Then the statement to create the MySQL account for `boris` looks like this:

```
CREATE USER 'boris'@'localhost'
  IDENTIFIED WITH authentication_ldap_sasl
  AS 'uid=boris_ldap,ou=People,dc=example,dc=com';
```

The authentication string specified in the `BY` clause does not include the LDAP password. That must be provided by the client user at connect time.

Clients connect to the MySQL server by providing the MySQL user name and LDAP password:

```
shell> mysql --user=boris --password
```
Enter password: **boris_password** *(boris_ldap LDAP password)*

For the server-side `authentication_ldap_sasl` plugin, clients use the client-side `authentication_ldap_sasl_client` plugin. If a client program does not find the client-side plugin, specify a `--plugin-dir` option that names the directory where the plugin library file is installed.

The authentication process for **boris** is similar to that previously described for **betsy** with simple LDAP authentication, except that the client-side and server-side SASL LDAP plugins use SASL messages for secure transmission of credentials within the LDAP protocol, to avoid sending the cleartext password between the MySQL client and server.

### LDAP Authentication with Proxying

LDAP authentication plugins support proxying, enabling a user to connect to the MySQL server as one user but assume the privileges of a different user. This section describes basic LDAP plugin proxy support. The LDAP plugins also support specification of group preference and proxy user mapping; see [LDAP Authentication Group Preference and Mapping Specification](#).

The proxying implementation described here is based on use of LDAP group attribute values to map connecting MySQL users who authenticate using LDAP onto other MySQL accounts that define different sets of privileges. Users do not connect directly through the accounts that define the privileges. Instead, they connect through a default proxy account authenticated with LDAP, such that all external logins are mapped to the proxied MySQL accounts that hold the privileges. Any user who connects using the proxy account is mapped to one of those proxied MySQL accounts, the privileges for which determine the database operations permitted to the external user.

The instructions here assume the following scenario:

- LDAP entries use the `uid` and `cn` attributes to specify user name and group values, respectively. To use different user and group attribute names, set the appropriate plugin-specific system variables:
  - For the `authentication_ldap_simple` plugin: Set `authentication_ldap_simple_user_search_attr` and `authentication_ldap_simple_group_search_attr`.
  - For the `authentication_ldap_sasl` plugin: Set `authentication_ldap_sasl_user_search_attr` and `authentication_ldap_sasl_group_search_attr`.

- These LDAP entries are available in the directory managed by the LDAP server, to provide distinguished name values that uniquely identify each user:
  
<table>
<thead>
<tr>
<th>Distinguished Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>uid=basha,ou=People,dc=example,dc=com,cn=accounting</td>
</tr>
<tr>
<td>uid=basil,ou=People,dc=example,dc=com,cn=front_office</td>
</tr>
</tbody>
</table>

  At connect time, the group attribute values become the authenticated user names, so they name the **accounting** and **front_office** proxied accounts.

- The examples assume use of SASL LDAP authentication. Make the appropriate adjustments for simple LDAP authentication.

Create the default proxy MySQL account:

```
CREATE USER ''@'%' IDENTIFIED WITH authentication_ldap_sasl;
```

The proxy account definition has no `AS 'auth_string'` clause to name an LDAP user DN. Thus:

- When a client connects, the client user name becomes the LDAP user name to search for.
- The matching LDAP entry is expected to include a group attribute naming the proxied MySQL account that defines the privileges the client should have.
Note

If your MySQL installation has anonymous users, they might conflict with the default proxy user. For more information about this issue, and ways of dealing with it, see Default Proxy User and Anonymous User Conflicts.

Create the proxied accounts and grant to each one the privileges it should have:

```
CREATE USER 'accounting'@'localhost'
IDENTIFIED WITH mysql_no_login;
CREATE USER 'front_office'@'localhost'
IDENTIFIED WITH mysql_no_login;
GRANT ALL PRIVILEGES
ON accountingdb.*
TO 'accounting'@'localhost';
GRANT ALL PRIVILEGES
ON frontdb.*
TO 'front_office'@'localhost';
```

The proxied accounts use the `mysql_no_login` authentication plugin to prevent clients from using the accounts to log in directly to the MySQL server. Instead, it is expected that users who authenticate using LDAP will use the default `''@'%'` proxy account. (This assumes that the `mysql_no_login` plugin is installed. For instructions, see Section 6.1.8, “No-Login Pluggable Authentication”.) For alternative methods of protecting proxied accounts against direct use, see Preventing Direct Login to Proxied Accounts.

Grant to the proxy account the `PROXY` privilege for each proxied account:

```
GRANT PROXY
ON 'accounting'@'localhost'
TO ''@'>';
GRANT PROXY
ON 'front_office'@'localhost'
TO ''@'>';
```

Use the `mysql` command-line client to connect to the MySQL server as `basha`.

```
shell> mysql --user=basha --password
Enter password: basha_password (basha LDAP password)
```

Authentication occurs as follows:

1. The server authenticates the connection using the default `''@'%'` proxy account, for client user `basha`.

2. The matching LDAP entry is:

```
uid=basha,ou=People,dc=example,dc=com,cn=accounting
```

3. The matching LDAP entry has group attribute `cn=accounting`, so `accounting` becomes the authenticated proxied user.

4. The authenticated user differs from the client user name `basha`, with the result that `basha` is treated as a proxy for `accounting`, and `basha` assumes the privileges of the proxied `accounting` account. The following query returns output as shown:

```
mysql> SELECT USER(), CURRENT_USER(), @@proxy_user;
+-----------------+----------------------+--------------+
| USER()          | CURRENT_USER()       | @@proxy_user |
+-----------------+----------------------+--------------+
| basha@localhost | accounting@localhost | ''@'%'       |
+-----------------+----------------------+--------------+
```

This demonstrates that `basha` uses the privileges granted to the proxied `accounting` MySQL account, and that proxying occurs through the default proxy user account.

Now connect as `basil` instead:
shell> mysql --user=basil --password
Enter password: basil_password (basil LDAP password)

The authentication process for basil is similar to that previously described for basha:

1. The server authenticates the connection using the default '@%' proxy account, for client user basil.

2. The matching LDAP entry is:
   
   uid=basil,ou=People,dc=example,dc=com,cn=front_office

3. The matching LDAP entry has group attribute cn=front_office, so front_office becomes the authenticated proxied user.

4. The authenticated user differs from the client user name basil, with the result that basil is treated as a proxy for front_office, and basil assumes the privileges of the proxied front_office account. The following query returns output as shown:

   mysql> SELECT USER(), CURRENT_USER(), @@proxy_user;
   +-----------------+------------------------+--------------+
   | USER()          | CURRENT_USER()         | @@proxy_user |
   +-----------------+------------------------+--------------+
   | basil@localhost | front_office@localhost | ''@'%'       |
   +-----------------+------------------------+--------------+

This demonstrates that basil uses the privileges granted to the proxied front_office MySQL account, and that proxying occurs through the default proxy user account.

### LDAP Authentication Group Preference and Mapping Specification

As described in LDAP Authentication with Proxying, basic LDAP authentication proxying works by the principle that the plugin uses the first group name returned by the LDAP server as the MySQL proxied user account name. This simple capability does not enable specifying any preference about which group name to use if the LDAP server returns multiple group names, or specifying any name other than the group name as the proxied user name.

As of MySQL 8.0.14, for MySQL accounts that use LDAP authentication, the authentication string can specify the following information to enable greater proxying flexibility:

- A list of groups in preference order, such that the plugin uses the first group name in the list that matches a group returned by the LDAP server.
- A mapping from group names to proxied user names, such that a group name when matched can provide a specified name to use as the proxied user. This provides an alternative to using the group name as the proxied user.

Consider the following MySQL proxy account definition:

```sql
CREATE USER ''@'%' IDENTIFIED WITH authentication_ldap_sasl
AS '+ou=People,dc=example,dc=com#grp1=usera,grp2,grp3=userc';
```

The authentication string has a user DN suffix `ou=People,dc=example,dc=com` prefixed by the `+` character. Thus, as described in LDAP Authentication User DN Suffixes, the full user DN is constructed from the user DN suffix as specified, plus the client user name as the `uid` attribute.

The remaining part of the authentication string begins with `#`, which signifies the beginning of group preference and mapping information. This part of the authentication string lists group names in the order `grp1, grp2, grp3`. The LDAP plugin compares that list with the set of group names returned by the LDAP server, looking in list order for a match against the returned names. The plugin uses the first match, or if there is no match, authentication fails.

Suppose that the LDAP server returns groups `grp3, grp2, grp7`. The LDAP plugin uses `grp2` because it is the first group in the authentication string that matches, even though it is not the first
LDAP Pluggable Authentication

group returned by the LDAP server. If the LDAP server returns grp4, grp2, and grp1, the plugin uses grp1 even though grp2 also matches. grp1 has a precedence higher than grp2 because it is listed earlier in the authentication string.

Assuming that the plugin finds a group name match, it performs mapping from that group name to the MySQL proxied user name, if there is one. For the example proxy account, mapping occurs as follows:

- If the matching group name is grp1 or grp3, those are associated in the authentication string with user names usera and userc, respectively. The plugin uses the corresponding associated user name as the proxied user name.
- If the matching group name is grp2, there is no associated user name in the authentication string. The plugin uses grp2 as the proxied user name.

If the LDAP server returns a group in DN format, the LDAP plugin parses the group DN to extract the group name from it.

To specify LDAP group preference and mapping information, these principles apply:

- Begin the group preference and mapping part of the authentication string with a # prefix character.
- The group preference and mapping specification is a list of one or more items, separated by commas. Each item has the form group_name=user_name or group_name. Items should be listed in group name preference order. For a group name selected by the plugin as a match from set of group names returned by the LDAP server, the two syntaxes differ in effect as follows:
  - For an item specified as group_name=user_name (with a user name), the group name maps to the user name, which is used as the MySQL proxied user name.
  - For an item specified as group_name (with no user name), the group name is used as the MySQL proxied user name.
- To quote a group or user name that contains special characters such as space, surround it by double quote (") characters. For example, if an item has group and user names of my group name and my user name, it must be written in a group mapping using quotes:

  "my group name"="my user name"

If an item has group and user names of my group name and my user name (which contain no special characters), it may but need not be written using quotes. Any of the following are valid:

  my_group_name=my_user_name
  my_group_name="my_user_name"
  "my_group_name"=my_user_name
  "my_group_name"="my_user_name"

- To escape a character, precede it by a backslash (\). This is useful particularly to include a literal double quote or backslash, which are otherwise not included literally.

- A user DN need not be present in the authentication string, but if present, it must precede the group preference and mapping part. A user DN can be given as a full user DN, or as a user DN suffix with a + prefix character. (See LDAP Authentication User DN Suffixes.)

LDAP Authentication User DN Suffixes

LDAP authentication plugins permit the authentication string that provides user DN information to begin with a + prefix character:

- In the absence of a + character, the authentication string value is treated as is without modification.
- If the authentication string begins with +, the plugin constructs the full user DN value from the user name sent by the client, together with the DN specified in the authentication string (with the + removed). In the constructed DN, the client user name becomes the value of the attribute
that specifies LDAP user names. This is uid by default; to change the attribute, modify the appropriate system variable (authentication_ldap_simple_user_search_attr or authentication_ldap_sasl_user_search_attr). The authentication string is stored as given in the mysql.user system table, with the full user DN constructed on the fly before authentication.

This account authentication string does not have + at the beginning, so it is taken as the full user DN:

```sql
CREATE USER 'baldwin'
    IDENTIFIED WITH authentication_ldap_simple
    AS 'uid=admin,ou=People,dc=example,dc=com';
```

The client connects with the user name specified in the account (baldwin). In this case, that name is not used because the authentication string has no prefix and thus fully specifies the user DN.

This account authentication string does have + at the beginning, so it is taken as just part of the user DN:

```sql
CREATE USER 'accounting'
    IDENTIFIED WITH authentication_ldap_simple
    AS '+ou=People,dc=example,dc=com';
```

The client connects with the user name specified in the account (accounting), which in this case is used as the uid attribute together with the authentication string to construct the user DN:

```sql
uid=accounting,ou=People,dc=example,dc=com
```

The accounts in the preceding examples have a nonempty user name, so the client always connects to the MySQL server using the same name as specified in the account definition. If an account has an empty user name, such as the default anonymous '''@''%' proxy account described in LDAP Authentication with Proxying, clients might connect to the MySQL server with varying user names. But the principle is the same: If the authentication string begins with +, the plugin uses the user name sent by the client together with the authentication string to construct the user DN.

**LDAP Authentication Methods**

The LDAP authentication plugins use a configurable authentication method. The appropriate system variable and available method choices are plugin-specific:

- For the authentication_ldap_simple plugin: Set the authentication_ldap_simple_auth_method_name system variable to configure the method. The permitted choices are SIMPLE and AD-FOREST.

- For the authentication_ldap_sasl plugin: Set the authentication_ldap_sasl_auth_method_name system variable to configure the method. The permitted choices are SCRAM-SHA-1 and GSSAPI. (To determine which SASL LDAP methods are actually available on the host system, check the value of the Authentication_ldap_sasl_supported_methods status variable.)

See the system variable descriptions for information about each permitted method. Also, depending on the method, additional configuration may be needed, as described in the following sections.

**The GSSAPI/Kerberos Authentication Method**

Generic Security Service Application Program Interface (GSSAPI) is a security abstraction interface. Kerberos is an instance of a specific security protocol that can be used through that abstract interface. Using GSSAPI, applications authenticate to Kerberos to obtain service credentials, then use those credentials in turn to enable secure access to other services.

One such service is LDAP, which is used by the client-side and server-side SASL LDAP authentication plugins. When the authentication_ldap_sasl_auth_method_name system variable is set to GSSAPI, these plugins use the GSSAPI/Kerberos authentication method. In this case, the plugins communicate securely using Kerberos without using LDAP messages directly. The server-side plugin then communicates with the LDAP server to interpret LDAP authentication messages and retrieve LDAP groups.
GSSAPI/Kerberos is supported as an authentication method for MySQL clients and servers only on Linux. It is useful in Linux environments where applications access LDAP using Microsoft Active Directory, which has Kerberos enabled by default.

The following discussion provides information about the configuration requirements for using the GSSAPI method. Familiarity is assumed with Kerberos concepts and operation, such as these common Kerberos terms:

• **Principal** = A named entity, such as a user or service.
• **KDC** = The Key Distribution Center, comprising the AS and TGS.
• **AS** = The Authentication Server, part of the KDC. Provides the initial ticket needed to obtain a TGT.
• **TGS** = The ticket-granting service, part of the KDC.
• **TGT** = The ticket-granting ticket, presented to the TGS to obtain service tickets for service access.

Kerberos authentication requires both a KDC server and an LDAP server. This requirement can be satisfied in different ways:

• Active Directory includes both servers, with Kerberos authentication enabled by default in the Active Directory LDAP server.
• OpenLDAP provides an LDAP server, but a separate KDC server may be needed, with additional Kerberos setup required.

Kerberos must also be available on the client host. A client contacts the AS using a password to obtain a TGT. The client then uses the TGT to obtain access from the TGS to other services, such as LDAP.

The following sections discuss the configuration steps to use GSSAPI/Kerberos for SASL LDAP authentication in MySQL:

• **Check the Kerberos Setup**
• **Configure the Server-Side SASL LDAP Authentication Plugin for GSSAPI/Kerberos**
• **Create a MySQL Account That Uses GSSAPI/Kerberos**
• **Use the MySQL Account to Connect to the MySQL Server**
• **/etc/krb5.conf Client Configuration Parameters**

**Check the Kerberos Setup**

The following example shows how to test availability of Kerberos in Active Directory. The example makes these assumptions:

• Active Directory is running on the host named `ldap_auth.example.com` with IP address `198.51.100.10`.
• The `MYSQL.LOCAL` domain is used for MySQL-related Kerberos authentication and LDAP lookups.
• A principal named `bredon@MYSQL.LOCAL` is registered with the KDC. (In later discussion, this principal name is also used for the MySQL user that uses GSSAPI/Kerberos to authenticate to the MySQL server.)

With those assumptions satisfied, follow this procedure:

1. Verify that the Kerberos library is installed and configured correctly in the operating system. For example, to configure a `MYSQL.LOCAL` domain for use during MySQL authentication, the Kerberos configuration file, `~/etc/krb5.conf`, should contain something like this:

```
[realms]
MYSQL.LOCAL = { 
```
2. You may need to add an entry to `/etc/hosts` for the server host:

```
198.51.100.10 ldap_auth ldap_auth.example.com
```

3. Check whether Kerberos authentication works correctly:
   a. Use `kinit` to authenticate to Kerberos:

   ```
kinit bredon@MYSQL.LOCAL
```
   The command authenticates for the Kerberos principal named `bredon@MYSQL.LOCAL`. Enter the principal's password when the command prompts for it. The KDC returns a TGT that is cached on the client side for use by other Kerberos-aware applications.

   b. Use `klist` to check whether the TGT was obtained correctly. The output should be similar to this:

   ```
Ticket cache: FILE:/tmp/krb5cc_244306
Default principal: bredon@MYSQL.LOCAL
Valid starting       Expires              Service principal
03/23/2020 08:18:33  03/23/2020 18:18:33  krbtgt/MYSQL.LOCAL@MYSQL.LOCAL
```

4. Check whether `ldapsearch` works with the Kerberos TGT (enter the command on a single line):

```
ldapsearch -h 198.51.100.10 -Y GSSAPI -D "cn=admin,cn=users,dc=MYSQL,dc=LOCAL" -b "dc=MYSQL,dc=LOCAL"
```

   The command searches for users in the `MYSQL.LOCAL` domain. In the `-D` option value, `admin` should be the name of an administrative LDAP account that has privileges to perform user searches.

**Configure the Server-Side SASL LDAP Authentication Plugin for GSSAPI/Kerberos**

Assuming that the LDAP server is accessible through Kerberos as just described, configure the server-side SASL LDAP authentication plugin to use the GSSAPI/Kerberos authentication method. (For general LDAP plugin installation information, see Installing LDAP Pluggable Authentication.) Here is an example of plugin-related settings the server `my.cnf` file might contain:

```
[mysqld]
plugin-load-add=authentication_ldap_sasl.so
authentication_ldap_sasl_auth_method_name="GSSAPI"
authentication_ldap_sasl_server_host=198.51.100.10
authentication_ldap_sasl_server_port=389
authentication_ldap_sasl_bind_root_dn="cn=admin,cn=users,dc=MYSQL,dc=LOCAL"
authentication_ldap_sasl_bind_root_pwd="password"
authentication_ldap_sasl_bind_base_dn="cn=users,dc=MYSQL,dc=LOCAL"
authentication_ldap_sasl_user_search_attr="sAMAccountName"
```

Those option file settings configure the SASL LDAP plugin as follows:

- The `--plugin-load-add` option loads the plugin (adjust the `.so` suffix for your platform as necessary). If you loaded the plugin previously using an `INSTALL PLUGIN` statement, this option is unnecessary.

- `authentication_ldap_sasl_auth_method_name` must be set to `GSSAPI` to use GSSAPI/Kerberos as the SASL LDAP authentication method.

- `authentication_ldap_sasl_server_host` and `authentication_ldap_sasl_server_port` indicate the IP address and port number of the Active Directory server host for authentication.
LDAP Pluggable Authentication

- `authentication_ldap_sasl_bind_root_dn` and `authentication_ldap_sasl_bind_root_pwd` configure the root DN and password for group search capability. This capability is required, but users may not have privileges to search. In such cases, it is necessary to provide root DN information:
  - In the DN option value, `admin` should be the name of an administrative LDAP account that has privileges to perform user searches.
  - In the password option value, `password` should be the `admin` account password.
- `authentication_ldap_sasl_bind_base_dn` indicates the user DN base path, so that searches look for users in the `MYSQL.LOCAL` domain.
- `authentication_ldap_sasl_user_search_attr` specifies a standard Active Directory search attribute, `sAMAccountName`. This attribute is used in searches to match logon names; attribute values are not the same as the user DN values.

Create a MySQL Account That Uses GSSAPI/Kerberos

MySQL authentication using the SASL LDAP authentication plugin with the GSSAPI/Kerberos method is based on a user that is a Kerberos principal. The following discussion uses a principal named `bredon@MYSQL.LOCAL` as this user, which must be registered in several places:

- The Kerberos administrator should register the user name as a Kerberos principal. This name should include a domain name. The principal name and password will be used by clients to authenticate with Kerberos and obtain a TGT.
- The LDAP administrator should register the user name in an LDAP entry. For example:
  ```
  uid=bredon,dc=MYSQL,dc=LOCAL
  ```
  **Note**
  In Active Directory (which uses Kerberos as the default authentication method), creating a user creates both the Kerberos principal and the LDAP entry.
- The MySQL DBA should create an account that has the Kerberos principal name as the user name, and that authenticates using the SASL LDAP plugin.

Assuming that the Kerberos principal and LDAP entry have been registered by the appropriate service administrators, and that the MySQL server has been started using the `my.cnf` settings previously described, create a MySQL account that corresponds to the Kerberos principal name, including the domain name.

**Note**

The SASL LDAP plugin uses a constant user DN for Kerberos authentication and ignores any user DN configured from MySQL. This has certain implications:

- For any MySQL account that uses GSSAPI/Kerberos authentication, the authentication string in `CREATE USER` or `ALTER USER` statements should contain no user DN because it has no effect.
- Because the authentication string contains no user DN, it should contain group mapping information, to enable the user to be handled as a proxy user that is mapped onto the desired proxied user. For information about proxying with the LDAP authentication plugin, see `LDAP Authentication with Proxying`.

The following statements create a proxy user named `bredon@MYSQL.LOCAL` that assumes the privileges of the proxied user named `proxied_krb_usr`. Other GSSAPI/Kerberos users that should have the same privileges can similarly be created as proxy users for the same proxied user.
-- create proxy account
CREATE USER 'bredon@MYSQL.LOCAL'
  IDENTIFIED WITH authentication_ldap_sasl
  BY 'krb_grp=proxied_krb_user';
-- create proxied account and grant its privileges;
-- use mysql_no_login plugin to prevent direct login
CREATE USER 'proxied_krb_user'
  IDENTIFIED WITH mysql_no_login;
GRANT ALL
  ON krb_user_db.*
TO 'proxied_krb_user';
-- grant to proxy account the
-- PROXY privilege for proxied account
GRANT PROXY
  ON 'proxied_krb_user'
TO 'bredon@MYSQL.LOCAL';

Observe closely the quoting for the proxy account name in the first CREATE USER statement and the
GRANT PROXY statement:

• For most MySQL accounts, the user and host are separate parts of the account name, and thus are quoted separately as 'user_name' '@' 'host_name'.

• For Kerberos authentication, the user part of the account name includes the principal domain, so 'bredon@MYSQL.LOCAL' is quoted as a single value. Because no host part is given, the full MySQL account name uses the default of '%' as the host part: 'bredon@MYSQL.LOCAL' '@' '%'

The proxied account uses the mysql_no_login authentication plugin to prevent clients from using the account to log in directly to the MySQL server. Instead, it is expected that users who authenticate using LDAP use the bredon@MYSQL.LOCAL proxy account. (This assumes that the mysql_no_login plugin is installed. For instructions, see Section 6.1.8, “No-Login Pluggable Authentication”.) For alternative methods of protecting proxied accounts against direct use, see Preventing Direct Login to Proxied Accounts.

Use the MySQL Account to Connect to the MySQL Server

After a MySQL account that uses GSSAPI/Kerberos has been set up, clients can authenticate to Kerberos and use the account to connect to the MySQL server. Kerberos authentication can take place either prior to or at the time of MySQL client program invocation:

• The client user can obtain a TGT independently of MySQL prior to invoking the MySQL client program. For example, the client user can use kinit to authenticate to Kerberos by providing a Kerberos principal name and the principal password. The TGT is cached and becomes available for use by other Kerberos-aware applications, such as the client-side SASL LDAP authentication plugin.

184
If the client command specifies no principal name as the user name and the client-side plugin finds the Kerberos cache empty because there is no TGT, authentication fails.

If you are uncertain whether a TGT exists, you can use `klist` to check.

Authentication occurs as follows:

1. The client uses the TGT to authenticate using Kerberos.
2. The server finds the LDAP entry for the principal and uses it to authenticate the connection for the `bredon@MYSQL.LOCAL` MySQL proxy account.
3. The group mapping information in the proxy account authentication string (`#krb_grp=proxied_krb_user`) indicates that the authenticated proxied user should be `proxied_krb_user`.
4. `bredon@MYSQL.LOCAL` is treated as a proxy for `proxied_krb_user`, and the following query returns output as shown:

   ```
   mysql> SELECT USER(), CURRENT_USER(), @@proxy_user;
   +------------------------------+--------------------+--------------------------+
   | USER()                       | CURRENT_USER()     | @@proxy_user             |
   +------------------------------+--------------------+--------------------------+
   | bredon@MYSQL.LOCAL@localhost | proxied_krb_user@% | 'bredon@MYSQL.LOCAL'@'%' |
   +------------------------------+--------------------+--------------------------+
   ```

   The `USER()` value indicates the user name used for the client command (`bredon@MYSQL.LOCAL` and the host from which the client connected (`localhost`).

   The `CURRENT_USER()` value is the full name of the proxied user account, which consists of the `proxied_krb_user` user part and the `%` host part.

   The `@@proxy_user` value indicates the full name of the account used to make the connection to the MySQL server, which consists of the `bredon@MYSQL.LOCAL` user part and the `%` host part.

   This demonstrates that proxying occurs through the `bredon@MYSQL.LOCAL` proxy user account, and that `bredon@MYSQL.LOCAL` assumes the privileges granted to the `proxied_krb_user` proxied user account.

A TGT once obtained is cached on the client side and can be used until it expires without specifying the password again. However the TGT is obtained, the client-side plugin uses it to acquire service tickets and communicate with the server-side plugin, without using LDAP directly.

When the client-side plugin itself obtains the TGT, the client user may not want the TGT to be reused. As described in `/etc/krb5.conf Client Configuration Parameters`, the local `/etc/krb5.conf` file can be used to cause the client-side plugin to destroy the TGT when done with it.

The server-side plugin has no access to the TGT itself or the Kerberos password used to obtain it.

The LDAP authentication plugins have no control over the caching mechanism (storage in a local file, in memory, and so forth), but Kerberos utilities such as `kswitch` may be available for this purpose.

/etc/krb5.conf Client Configuration Parameters

The client-side SASL LDAP plugin reads the local `/etc/krb5.conf` file. If this file is missing or inaccessible, an error occurs. Assuming that the file is accessible, the optional `[appdefaults]` group can be used to provide information used by the plugin. Place such information within the `MySQL` section of the group. For example:
No-Login Pluggable Authentication

[appdefaults]
MySQL = {
    ldap_server_host = "ldap_host.example.com"
    ldap_destroy_tgt = true
}

The client-side plugin recognizes these parameters in the MySQL section:

- The ldap_server_host value specifies the LDAP server host and can be useful when that host differs from the KDC server host specified in the [realms] group. By default, the plugin uses the KDC server host as the LDAP server host.
- The ldap_destroy_tgt value indicates whether the client-side plugin destroys the TGT after obtaining and using it. By default, ldap_destroy_tgt is false, but can be set to true to avoid TGT reuse. (This setting applies only to TGTs created by the client-side plugin, not TGTs created externally to MySQL.)

LDAP Search Referral

An LDAP server can be configured to delegate LDAP searches to another LDAP server, a functionality known as LDAP referral. Suppose that the server a.example.com holds a "dc=example,dc=com" root DN and wishes to delegate searches to another server b.example.com. To enable this, a.example.com would be configured with a named referral object having these attributes:

<table>
<thead>
<tr>
<th>dn</th>
<th>dc=subtree,dc=example,dc=com</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectClass</td>
<td>referral</td>
</tr>
<tr>
<td>objectClass</td>
<td>extensibleObject</td>
</tr>
<tr>
<td>dc</td>
<td>subtree</td>
</tr>
<tr>
<td>ref</td>
<td>ldap://b.example.com/dc=subtree,dc=example,dc=com</td>
</tr>
</tbody>
</table>

An issue with enabling LDAP referral is that searches can fail with LDAP operation errors when the search base DN is the root DN, and referral objects are not set. A MySQL DBA might wish to avoid such referral errors for the LDAP authentication plugins, even though LDAP referral might be set globally in the ldap.conf configuration file. To configure on a plugin-specific basis whether the LDAP server should use LDAP referral when communicating with each plugin, set the authentication_ldap_simple_referral and authentication_ldap_sasl_referral system variables. Setting either variable to ON or OFF causes the corresponding LDAP authentication plugin to tell the LDAP server whether to use referral during MySQL authentication. Each variable has a plugin-specific effect and does not affect other applications that communicate with the LDAP server. Both variables are OFF by default.

6.1.8 No-Login Pluggable Authentication

The mysql_no_login server-side authentication plugin prevents all client connections to any account that uses it. Use cases for this plugin include:

- Accounts that must be able to execute stored programs and views with elevated privileges without exposing those privileges to ordinary users.
- Proxied accounts that should never permit direct login but are intended to be accessed only through proxy accounts.

The following table shows the plugin and library file names. The file name suffix might differ on your system. The file must be located in the directory named by the plugin_dir system variable.

<table>
<thead>
<tr>
<th>Plugin or File</th>
<th>Plugin or File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server-side plugin</td>
<td>mysql_no_login</td>
</tr>
<tr>
<td>Client-side plugin</td>
<td>None</td>
</tr>
<tr>
<td>Library file</td>
<td>mysql_no_login.so</td>
</tr>
</tbody>
</table>
The following sections provide installation and usage information specific to no-login pluggable authentication:

- Installing No-Login Pluggable Authentication
- Uninstalling No-Login Pluggable Authentication
- Using No-Login Pluggable Authentication

For general information about pluggable authentication in MySQL, see Section 4.17, “Pluggable Authentication”. For proxy user information, see Section 4.18, “Proxy Users”.

**Installing No-Login Pluggable Authentication**

This section describes how to install the no-login authentication plugin. For general information about installing plugins, see Installing and Uninstalling Plugins.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

The plugin library file base name is `mysql_no_login`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

To load the plugin at server startup, use the `--plugin-load-add` option to name the library file that contains it. With this plugin-loading method, the option must be given each time the server starts. For example, put these lines in the server `my.cnf` file (adjust the `.so` suffix for your platform as necessary):

```
[mysqld]
plugin-load-add=mysql_no_login.so
```

After modifying `my.cnf`, restart the server to cause the new settings to take effect.

Alternatively, to load the plugin at runtime, use this statement (adjust the `.so` suffix for your platform as necessary):

```
INSTALL PLUGIN mysql_no_login SONAME 'mysql_no_login.so';
```

`INSTALL PLUGIN` loads the plugin immediately, and also registers it in the `mysql.plugins` system table to cause the server to load it for each subsequent normal startup without the need for `--plugin-load-add`.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement (see Obtaining Server Plugin Information). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
FROM INFORMATION_SCHEMA.PLUGINS
WHERE PLUGIN_NAME LIKE '%login%';
```

If the plugin fails to initialize, check the server error log for diagnostic messages.

To associate MySQL accounts with the no-login plugin, see Using No-Login Pluggable Authentication.

**Uninstalling No-Login Pluggable Authentication**

The method used to uninstall the no-login authentication plugin depends on how you installed it:

- If you installed the plugin at server startup using a `--plugin-load-add` option, restart the server without the option.
No-Login Pluggable Authentication

• If you installed the plugin at runtime using an `INSTALL PLUGIN` statement, it remains installed across server restarts. To uninstall it, use `UNINSTALL PLUGIN`:

  UNINSTALL PLUGIN mysql_no_login;

Using No-Login Pluggable Authentication

This section describes how to use the no-login authentication plugin to prevent accounts from being used for connecting from MySQL client programs to the server. It is assumed that the server is running with the no-login plugin enabled, as described in Installing No-Login Pluggable Authentication.

To refer to the no-login authentication plugin in the `IDENTIFIED WITH` clause of a `CREATE USER` statement, use the name `mysql_no_login`.

An account that authenticates using `mysql_no_login` may be used as the `DEFINER` for stored program and view objects. If such an object definition also includes `SQL SECURITY DEFINER`, it executes with that account's privileges. DBAs can use this behavior to provide access to confidential or sensitive data that is exposed only through well-controlled interfaces.

The following example illustrates these principles. It defines an account that does not permit client connections, and associates with it a view that exposes only certain columns of the `mysql.user` system table:

```
CREATE DATABASE nologindb;
CREATE USER 'nologin'@'localhost'
  IDENTIFIED WITH mysql_no_login;
GRANT ALL ON nologindb.*
  TO 'nologin'@'localhost';
GRANT SELECT ON mysql.user
  TO 'nologin'@'localhost';
CREATE DEFINER = 'nologin'@'localhost'
  SQL SECURITY DEFINER
  VIEW nologindb.myview
  AS SELECT User, Host FROM mysql.user;
```

To provide protected access to the view to an ordinary user, do this:

```
GRANT SELECT ON nologindb.myview
  TO 'ordinaryuser'@'localhost';
```

Now the ordinary user can use the view to access the limited information it presents:

```
SELECT * FROM nologindb.myview;
```

Attempts by the user to access columns other than those exposed by the view result in an error, as do attempts to select from the view by users not granted access to it.

---

Note

Because the `nologin` account cannot be used directly, the operations required to set up objects that it uses must be performed by `root` or similar account that has the privileges required to create the objects and set `DEFINER` values.

The `mysql_no_login` plugin is also useful in proxying scenarios. (For a discussion of concepts involved in proxying, see Section 4.18, “Proxy Users”.) An account that authenticates using `mysql_no_login` may be used as a proxied user for proxy accounts:

```
-- create proxied account
CREATE USER 'proxied_user'@'localhost'
  IDENTIFIED WITH mysql_no_login;
-- grant privileges to proxied account
GRANT ...
  ON ...
  TO 'proxied_user'@'localhost';
-- permit proxy_user to be a proxy account for proxied account
```
This enables clients to access MySQL through the proxy account (proxy_user) but not to bypass the proxy mechanism by connecting directly as the proxied user (proxied_user). A client who connects using the proxy_user account has the privileges of the proxied_user account, but proxied_user itself cannot be used to connect.

For alternative methods of protecting proxied accounts against direct use, see Preventing Direct Login to Proxied Accounts.

6.1.9 Socket Peer-Credential Pluggable Authentication

The server-side auth_socket authentication plugin authenticates clients that connect from the local host through the Unix socket file. The plugin uses the SO_PEERCRED socket option to obtain information about the user running the client program. Thus, the plugin can be used only on systems that support the SO_PEERCRED option, such as Linux.

The source code for this plugin can be examined as a relatively simple example demonstrating how to write a loadable authentication plugin.

The following table shows the plugin and library file names. The file must be located in the directory named by the plugin_dir system variable.

<table>
<thead>
<tr>
<th>Plugin or File</th>
<th>Plugin or File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server-side plugin</td>
<td>auth_socket</td>
</tr>
<tr>
<td>Client-side plugin</td>
<td>None, see discussion</td>
</tr>
<tr>
<td>Library file</td>
<td>auth_socket.so</td>
</tr>
</tbody>
</table>

The following sections provide installation and usage information specific to socket pluggable authentication:

- Installing Socket Pluggable Authentication
- Uninstalling Socket Pluggable Authentication
- Using Socket Pluggable Authentication

For general information about pluggable authentication in MySQL, see Section 4.17, “Pluggable Authentication”.

Installing Socket Pluggable Authentication

This section describes how to install the socket authentication plugin. For general information about installing plugins, see Installing and Uninstalling Plugins.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the plugin_dir system variable). If necessary, configure the plugin directory location by setting the value of plugin_dir at server startup.

To load the plugin at server startup, use the --plugin-load-add option to name the library file that contains it. With this plugin-loading method, the option must be given each time the server starts. For example, put these lines in the server my.cnf file:

```
[mysqld]
plugin-load-add=auth_socket.so
```

After modifying my.cnf, restart the server to cause the new settings to take effect.
Alternatively, to load the plugin at runtime, use this statement:

```
INSTALL PLUGIN auth_socket SONAME 'auth_socket.so';
```

`INSTALL PLUGIN` loads the plugin immediately, and also registers it in the `mysql.plugins` system table to cause the server to load it for each subsequent normal startup without the need for `--plugin-load-add`.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement (see Obtaining Server Plugin Information). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
    FROM INFORMATION_SCHEMA.PLUGINS
    WHERE PLUGIN_NAME LIKE '%socket%';
```

<table>
<thead>
<tr>
<th>PLUGIN_NAME</th>
<th>PLUGIN_STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>auth_socket</td>
<td>ACTIVE</td>
</tr>
</tbody>
</table>

If the plugin fails to initialize, check the server error log for diagnostic messages.

To associate MySQL accounts with the socket plugin, see Using Socket Pluggable Authentication.

### Uninstalling Socket Pluggable Authentication

The method used to uninstall the socket authentication plugin depends on how you installed it:

- If you installed the plugin at server startup using a `--plugin-load-add` option, restart the server without the option.
- If you installed the plugin at runtime using an `INSTALL PLUGIN` statement, it remains installed across server restarts. To uninstall it, use `UNINSTALL PLUGIN`:

```
UNINSTALL PLUGIN auth_socket;
```

### Using Socket Pluggable Authentication

The socket plugin checks whether the socket user name (the operating system user name) matches the MySQL user name specified by the client program to the server. If the names do not match, the plugin checks whether the socket user name matches the name specified in the `authentication_string` column of the `mysql.user` system table row. If a match is found, the plugin permits the connection. The `authentication_string` value can be specified using an `IDENTIFIED ...AS` clause with `CREATE USER` or `ALTER USER`.

Suppose that a MySQL account is created for an operating system user named `valerie` who is to be authenticated by the `auth_socket` plugin for connections from the local host through the socket file:

```
CREATE USER 'valerie'@'localhost' IDENTIFIED WITH auth_socket;
```

If a user on the local host with a login name of `stefanie` invokes `mysql` with the option `--user=valerie` to connect through the socket file, the server uses `auth_socket` to authenticate the client. The plugin determines that the `--user` option value (`valerie`) differs from the client user's name (`stephanie`) and refuses the connection. If a user named `valerie` tries the same thing, the plugin finds that the user name and the MySQL user name are both `valerie` and permits the connection. However, the plugin refuses the connection even for `valerie` if the connection is made using a different protocol, such as TCP/IP.

To permit both the `valerie` and `stephanie` operating system users to access MySQL through socket file connections that use the account, this can be done two ways:

- Name both users at account-creation time, one following `CREATE USER`, and the other in the authentication string:
CREATE USER 'valerie'@'localhost' IDENTIFIED WITH auth_socket AS 'stephanie';

- If you have already used `CREATE USER` to create the account for a single user, use `ALTER USER` to add the second user:

  CREATE USER 'valerie'@'localhost' IDENTIFIED WITH auth_socket;
  ALTER USER 'valerie'@'localhost' IDENTIFIED WITH auth_socket AS 'stephanie';

To access the account, both `valerie` and `stephanie` specify `--user=valerie` at connect time.

## 6.1.10 Test Pluggable Authentication

MySQL includes a test plugin that checks account credentials and logs success or failure to the server error log. This is a loadable plugin (not built in) and must be installed prior to use.

The test plugin source code is separate from the server source, unlike the built-in native plugin, so it can be examined as a relatively simple example demonstrating how to write a loadable authentication plugin.

**Note**

This plugin is intended for testing and development purposes, and is not for use in production environments or on servers that are exposed to public networks.

The following table shows the plugin and library file names. The file name suffix might differ on your system. The file must be located in the directory named by the `plugin_dir` system variable.

### Table 6.11 Plugin and Library Names for Test Authentication

<table>
<thead>
<tr>
<th>Plugin or File</th>
<th>Plugin or File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server-side plugin</td>
<td>test_plugin_server</td>
</tr>
<tr>
<td>Client-side plugin</td>
<td>auth_test_plugin</td>
</tr>
<tr>
<td>Library file</td>
<td>auth_test_plugin.so</td>
</tr>
</tbody>
</table>

The following sections provide installation and usage information specific to test pluggable authentication:

- Installing Test Pluggable Authentication
- Uninstalling Test Pluggable Authentication
- Using Test Pluggable Authentication

For general information about pluggable authentication in MySQL, see [Section 4.17, “Pluggable Authentication”](#).

### Installing Test Pluggable Authentication

This section describes how to install the test authentication plugin. For general information about installing plugins, see [Installing and Uninstalling Plugins](#).

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

To load the plugin at server startup, use the `--plugin-load-add` option to name the library file that contains it. With this plugin-loading method, the option must be given each time the server starts. For example, put these lines in the server `my.cnf` file (adjust the `.so` suffix for your platform as necessary):

```sql
[mysqld]
plugin_dir = /path/to/plugin
```
Test Pluggable Authentication

```sql
[mysqld]
plugin-load-add=auth_test_plugin.so
```

After modifying `my.cnf`, restart the server to cause the new settings to take effect.

Alternatively, to load the plugin at runtime, use this statement (adjust the `.so` suffix for your platform as necessary):

```sql
INSTALL PLUGIN test_plugin_server SONAME 'auth_test_plugin.so';
```

`INSTALL PLUGIN` loads the plugin immediately, and also registers it in the `mysql.plugins` system table to cause the server to load it for each subsequent normal startup without the need for `--plugin-load-add`.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement (see `Obtaining Server Plugin Information`). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
FROM INFORMATION_SCHEMA.PLUGINS
WHERE PLUGIN_NAME LIKE '%test_plugin%';
+--------------------+---------------+
| PLUGIN_NAME        | PLUGIN_STATUS |
+--------------------+---------------+
| test_plugin_server | ACTIVE        |
+--------------------+---------------+
```

If the plugin fails to initialize, check the server error log for diagnostic messages.

To associate MySQL accounts with the test plugin, see `Using Test Pluggable Authentication`.

### Uninstalling Test Pluggable Authentication

The method used to uninstall the test authentication plugin depends on how you installed it:

- If you installed the plugin at server startup using a `--plugin-load-add` option, restart the server without the option.

- If you installed the plugin at runtime using an `INSTALL PLUGIN` statement, it remains installed across server restarts. To uninstall it, use `UNINSTALL PLUGIN`:

```
UNINSTALL PLUGIN test_plugin_server;
```

### Using Test Pluggable Authentication

To use the test authentication plugin, create an account and name that plugin in the `IDENTIFIED WITH` clause:

```sql
CREATE USER 'testuser'@'localhost'
IDENTIFIED WITH test_plugin_server
BY 'testpassword';
```

Then provide the `--user` and `--password` options for that account when you connect to the server. For example:

```
shell> mysql --user=testuser --password
Enter password: testpassword
```

The plugin fetches the password as received from the client and compares it with the value stored in the `authentication_string` column of the account row in the `mysql.user` system table. If the two values match, the plugin returns the `authentication_string` value as the new effective user ID.

You can look in the server error log for a message indicating whether authentication succeeded (notice that the password is reported as the “user”):

```
[Note] Plugin test_plugin_server reported:
```
6.1.11 Pluggable Authentication System Variables

These variables are unavailable unless the appropriate server-side plugin is installed:

- `authentication_ldap_sasl` for system variables with names of the form `authentication_ldap_sasl_xxx`
- `authentication_ldap_simple` for system variables with names of the form `authentication_ldap_simple_xxx`

Table 6.12 Authentication Plugin System Variable Summary

<table>
<thead>
<tr>
<th>Name</th>
<th>Cmd-Line</th>
<th>Option File</th>
<th>System Var</th>
<th>Status Var</th>
<th>Var Scope</th>
<th>Dyn</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>authentication_ldap_sasl_auth_method_name</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td><code>authentication_ldap_sasl_bind_base_dn</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td><code>authentication_ldap_sasl_bind_root_dn</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td><code>authentication_ldap_sasl_bind_root_pwd</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td><code>authentication_ldap_sasl_cacert</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td><code>authentication_ldap_sasl_group_search_attr</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td><code>authentication_ldap_sasl_group_search_filter</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td><code>authentication_ldap_sasl_init_pool_size</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td><code>authentication_ldap_sasl_log_status</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td><code>authentication_ldap_sasl_max_pool_size</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td><code>authentication_ldap_sasl_referral</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td><code>authentication_ldap_sasl_server_host</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td><code>authentication_ldap_sasl_server_port</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td><code>authentication_ldap_sasl_tls</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td><code>authentication_ldap_simple_auth_method_name</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td><code>authentication_ldap_simple_bind_base_dn</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td><code>authentication_ldap_simple_bind_root_dn</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td><code>authentication_ldap_simple_bind_root_pwd</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td><code>authentication_ldap_simple_cacert</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td><code>authentication_ldap_simple_group_search_attr</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td><code>authentication_ldap_simple_group_search_filter</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td><code>authentication_ldap_simple_init_pool_size</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td><code>authentication_ldap_simple_log_status</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td><code>authentication_ldap_simple_max_pool_size</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td><code>authentication_ldap_simple_referral</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td><code>authentication_ldap_simple_server_host</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td><code>authentication_ldap_simple_server_port</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td><code>authentication_ldap_simple_tls</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Global</td>
<td>Yes</td>
</tr>
<tr>
<td><code>authentication_windows_log_level</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Global</td>
<td>No</td>
</tr>
<tr>
<td><code>authentication_windows_use_principal_name</code></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Global</td>
<td>No</td>
</tr>
</tbody>
</table>
**Pluggable Authentication System Variables**

- **authentication_ldap_sasl_auth_method_name**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--authentication-ldap-sasl-auth-method-name=value</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>authentication_ldap_sasl_auth_method_name</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>SET_VAR</strong> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td><code>SCRAM-SHA-1</code></td>
</tr>
<tr>
<td>Valid Values (<code>≥ 8.0.20</code>)</td>
<td><code>SCRAM-SHA-1</code></td>
</tr>
<tr>
<td>Valid Values (<code>≤ 8.0.19</code>)</td>
<td><code>SCRAM-SHA-1</code>, <code>GSSAPI</code></td>
</tr>
</tbody>
</table>

For SASL LDAP authentication, the authentication method name. Communication between the authentication plugin and the LDAP server occurs according to this authentication method to ensure password security.

These authentication method values are permitted:

- **SCRAM-SHA-1**: Use a SASL challenge-response mechanism.

  The client-side `authentication_ldap_sasl_client` plugin communicates with the SASL server, using the password to create a challenge and obtain a SASL request buffer, then passes this buffer to the server-side `authentication_ldap_sasl` plugin. The client-side and server-side SASL LDAP plugins use SASL messages for secure transmission of credentials within the LDAP protocol, to avoid sending the cleartext password between the MySQL client and server.

- **GSSAPI**: Use Kerberos, a passwordless and ticket-based protocol.

  GSSAPI/Kerberos is supported as an authentication method for MySQL clients and servers only on Linux. It is useful in Linux environments where applications access LDAP using Microsoft Active Directory, which has Kerberos enabled by default.

  The client-side `authentication_ldap_sasl_client` plugin obtains a service ticket using the ticket-granting ticket (TGT) from Kerberos, but does not use LDAP services directly. The server-side `authentication_ldap_sasl` plugin routes Kerberos messages between the client-side plugin and the LDAP server. Using the credentials thus obtained, the server-side plugin then communicates with the LDAP server to interpret LDAP authentication messages and retrieve LDAP groups.

- **authentication_ldap_sasl_bind_base_dn**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--authentication-ldap-sasl-bind-base-dn=value</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>authentication_ldap_sasl_bind_base_dn</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>SET_VAR</strong> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
</tbody>
</table>
### Pluggable Authentication System Variables

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>NULL</td>
</tr>
</tbody>
</table>

For SASL LDAP authentication, the base distinguished name (DN). This variable can be used to limit the scope of searches by anchoring them at a certain location (the “base”) within the search tree.

Suppose that members of one set of LDAP user entries each have this form:

```
uid=user_name,ou=People,dc=example,dc=com
```

And that members of another set of LDAP user entries each have this form:

```
uid=user_name,ou=Admin,dc=example,dc=com
```

Then searches work like this for different base DN values:

- If the base DN is `ou=People,dc=example,dc=com`: Searches find user entries only in the first set.
- If the base DN is `ou=Admin,dc=example,dc=com`: Searches find user entries only in the second set.
- If the base DN is `ou=dc=example,dc=com`: Searches find user entries in the first or second set.

In general, more specific base DN values result in faster searches because they limit the search scope more.

#### `authentication_ldap_sasl_bind_root_dn`

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--authentication-ldap-sasl-bind-root-dn=value</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>authentication_ldap_sasl_bind_root_dn</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td>NULL</td>
</tr>
</tbody>
</table>

For SASL LDAP authentication, the root distinguished name (DN). This variable is used in conjunction with `authentication_ldap_sasl_bind_root_pwd` as the credentials for authenticating to the LDAP server for the purpose of performing searches. Authentication uses either one or two LDAP bind operations, depending on whether the MySQL account names an LDAP user DN:

- If the account does not name a user DN: `authentication_ldap_sasl` performs an initial LDAP binding using `authentication_ldap_sasl_bind_root_dn` and `authentication_ldap_sasl_bind_root_pwd`. (These are both empty by default, so if they are not set, the LDAP server must permit anonymous connections.) The resulting bind LDAP handle is used to search for the user DN, based on the client user name. `authentication_ldap_sasl` performs a second bind using the user DN and client-supplied password.
- If the account does name a user DN: The first bind operation is unnecessary in this case. `authentication_ldap_sasl` performs a single bind using the user DN and client-supplied password. This is faster than if the MySQL account does not specify an LDAP user DN.
**Pluggable Authentication System Variables**

- **authentication_ldap_sasl_bind_root_pwd**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td>--authentication-ldap-sasl-bind-root-pwd=value</td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td>authentication_ldap_sasl_bind_root_pwd</td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>SET_VAR</strong> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td>NULL</td>
</tr>
</tbody>
</table>

For SASL LDAP authentication, the password for the root distinguished name. This variable is used in conjunction with `authentication_ldap_sasl_bind_root_dn`. See the description of that variable.

- **authentication_ldap_sasl_ca_path**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td>--authentication-ldap-sasl-ca-path=value</td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td>authentication_ldap_sasl_ca_path</td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>SET_VAR</strong> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td>NULL</td>
</tr>
</tbody>
</table>

For SASL LDAP authentication, the absolute path of the certificate authority file. Specify this file if it is desired that the authentication plugin perform verification of the LDAP server certificate.

| Note | In addition to setting the `authentication_ldap_sasl_ca_path` variable to the file name, you must add the appropriate certificate authority certificates to the file and enable the `authentication_ldap_sasl_tls` system variable. |

- **authentication_ldap_sasl_group_search_attr**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td>--authentication-ldap-sasl-group-search-attr=value</td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td>authentication_ldap_sasl_group_search_attr</td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>SET_VAR</strong> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
</tbody>
</table>


For SASL LDAP authentication, the name of the attribute that specifies group names in LDAP directory entries. If `authentication_ldap_sasl_group_search_attr` has its default value of `cn`, searches return the `cn` value as the group name. For example, if an LDAP entry with a `uid` value of `user1` has a `cn` attribute of `mygroup`, searches for `user1` return `mygroup` as the group name.

This variable should be the empty string if you want no group or proxy authentication.

If the group search attribute is `isMemberOf`, LDAP authentication directly retrieves the user attribute `isMemberOf` value and assigns it as group information. If the group search attribute is not `isMemberOf`, LDAP authentication searches for all groups where the user is a member. (The latter is the default behavior.) This behavior is based on how LDAP group information can be stored two ways: 1) A group entry can have an attribute named `memberUid` or `member` with a value that is a user name; 2) A user entry can have an attribute named `isMemberOf` with values that are group names.

• `authentication_ldap_sasl_group_search_filter`

For SASL LDAP authentication, the custom group search filter.

The search filter value can contain `{UA}` and `{UD}` notation to represent the user name and the full user DN. For example, `{UA}` is replaced with a user name such as "admin", whereas `{UD}` is replaced with a use full DN such as "uid=admin,ou=People,dc=example,dc=com". The following value is the default, which supports both OpenLDAP and Active Directory:

```
( |(&(objectClass=posixGroup)(memberUid={UA}))
 |(&(objectClass=group)(member={UD})))
```

In some cases for the user scenario, `memberOf` is a simple user attribute that holds no group information. For additional flexibility, an optional `{GA}` prefix can be used with the group search attribute. Any group attribute with a `{GA}` prefix is treated as a user attribute having group names. For example, with a value of `{GA}MemberOf`, if the group value is the DN, the first attribute value from the group DN is returned as the group name.

• `authentication_ldap_sasl_init_pool_size`

For SASL LDAP authentication, the custom group search filter.
For SASL LDAP authentication, the initial size of the pool of connections to the LDAP server. Choose the value for this variable based on the average number of concurrent authentication requests to the LDAP server.

The plugin uses `authentication_ldap_sasl_init_pool_size` and `authentication_ldap_sasl_max_pool_size` together for connection-pool management:

- When the authentication plugin initializes, it creates `authentication_ldap_sasl_init_pool_size` connections, unless `authentication_ldap_sasl_max_pool_size=0` to disable pooling.

- If the plugin receives an authentication request when there are no free connections in the current connection pool, the plugin can create a new connection, up to the maximum connection pool size given by `authentication_ldap_sasl_max_pool_size`.

- If the plugin receives a request when the pool size is already at its maximum and there are no free connections, authentication fails.

- When the plugin unloads, it closes all pooled connections.

Changes to plugin system variable settings may have no effect on connections already in the pool. For example, modifying the LDAP server host, port, or TLS settings does not affect existing connections. However, if the original variable values were invalid and the connection pool could not be initialized, the plugin attempts to reinitialize the pool for the next LDAP request. In this case, the new system variable values are used for the reinitialization attempt.

If `authentication_ldap_sasl_max_pool_size=0` to disable pooling, each LDAP connection opened by the plugin uses the values the system variables have at that time.

- `authentication_ldap_sasl_log_status`
For SASL LDAP authentication, the logging level for messages written to the error log. The following table shows the permitted level values and their meanings.

### Table 6.13 Log Levels for `authentication_ldap_sasl_log_status`

<table>
<thead>
<tr>
<th>Option Value</th>
<th>Types of Messages Logged</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>No messages</td>
</tr>
<tr>
<td>2</td>
<td>Error messages</td>
</tr>
<tr>
<td>3</td>
<td>Error and warning messages</td>
</tr>
<tr>
<td>4</td>
<td>Error, warning, and information messages</td>
</tr>
<tr>
<td>5</td>
<td>Same as previous level plus debugging messages from MySQL</td>
</tr>
<tr>
<td>6</td>
<td>Same as previous level plus debugging messages from LDAP library</td>
</tr>
</tbody>
</table>

Log level 6 is available as of MySQL 8.0.18.

On the client side, messages can be logged to the standard output by setting the `AUTHENTICATION_LDAP_CLIENT_LOG` environment variable. The permitted and default values are the same as for `authentication_ldap_sasl_log_status`.

The `AUTHENTICATION_LDAP_CLIENT_LOG` environment variable applies only to SASL LDAP authentication. It has no effect for simple LDAP authentication because the client plugin in that case is `mysql_clear_password`, which knows nothing about LDAP operations.

- `authentication_ldap_sasl_max_pool_size`

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Command-Line Format</strong></td>
<td><code>--authentication-ldap-sasl-max-pool-size=#</code></td>
</tr>
<tr>
<td><strong>Introduced</strong></td>
<td>8.0.11</td>
</tr>
<tr>
<td><strong>System Variable</strong></td>
<td><code>authentication_ldap_sasl_max_pool_size</code></td>
</tr>
<tr>
<td><strong>Scope</strong></td>
<td>Global</td>
</tr>
<tr>
<td><strong>SET_VAR Hint Applies</strong></td>
<td>No</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>Integer</td>
</tr>
<tr>
<td><strong>Default Value</strong></td>
<td>1000</td>
</tr>
<tr>
<td><strong>Minimum Value</strong></td>
<td>0</td>
</tr>
<tr>
<td><strong>Maximum Value</strong></td>
<td>32767</td>
</tr>
</tbody>
</table>

For SASL LDAP authentication, the maximum size of the pool of connections to the LDAP server. To disable connection pooling, set this variable to 0.

This variable is used in conjunction with `authentication_ldap_sasl_init_pool_size`. See the description of that variable.

- `authentication_ldap_sasl_referral`

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Command-Line Format</strong></td>
<td>`--authentication-ldap-sasl-referral=[OFF</td>
</tr>
<tr>
<td><strong>Introduced</strong></td>
<td>8.0.20</td>
</tr>
</tbody>
</table>
### Pluggable Authentication System Variables

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Variable</td>
<td><code>authentication_ldap_sasl_referral</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>SET_VAR</strong> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>OFF</td>
</tr>
</tbody>
</table>

For SASL LDAP authentication, whether to enable LDAP search referral. See [LDAP Search Referral](#).

- `authentication_ldap_sasl_server_host`

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--authentication-ldap-sasl-server-host=host_name</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>authentication_ldap_sasl_server_host</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>SET_VAR</strong> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
</tbody>
</table>

For SASL LDAP authentication, the LDAP server host. The permitted values for this variable depend on the authentication method:

- For `authentication_ldap_sasl_auth_method_name=SCRAM-SHA-1`: The LDAP server host can be a host name or IP address.

- `authentication_ldap_sasl_server_port`

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--authentication-ldap-sasl-server-port=port_num</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>authentication_ldap_sasl_server_port</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>SET_VAR</strong> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>389</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>1</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>32376</td>
</tr>
</tbody>
</table>

For SASL LDAP authentication, the LDAP server TCP/IP port number.

As of MySQL 8.0.14, if the LDAP port number is configured as 636 or 3269, the plugin uses LDAPS (LDAP over SSL) instead of LDAP. (LDAPS differs from `startTLS`.)
Pluggable Authentication System Variables

- **authentication_ldap_sasl_tls**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td>`--authentication-ldap-sasl-tls=[{OFF</td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>authentication_ldap_sasl_tls</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>OFF</td>
</tr>
</tbody>
</table>

For SASL LDAP authentication, whether connections by the plugin to the LDAP server are secure. If this variable is enabled, the plugin uses TLS to connect securely to the LDAP server. If you enable this variable, you may also wish to set the `authentication_ldap_sasl_ca_path` variable.

MySQL LDAP plugins support the StartTLS method, which initializes TLS on top of a plain LDAP connection. The `ldaps` method is deprecated and MySQL does not support it.

- **authentication_ldap_sasl_user_search_attr**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--authentication-ldap-sasl-user-search-attr=value</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>authentication_ldap_sasl_user_search_attr</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td><code>uid</code></td>
</tr>
</tbody>
</table>

For SASL LDAP authentication, the name of the attribute that specifies user names in LDAP directory entries. If a user distinguished name is not provided, the authentication plugin searches for the name using this attribute. For example, if the `authentication_ldap_sasl_user_search_attr` value is `uid`, a search for the user name `user1` finds entries with a `uid` value of `user1`.

- **authentication_ldap_simple_auth_method_name**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--authentication-ldap-simple-auth-method-name=value</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>authentication_ldap_simple_auth_method_name</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Valid Values</td>
<td>SIMPLE</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td><code>SIMPLE</code></td>
</tr>
</tbody>
</table>
For simple LDAP authentication, the authentication method name. Communication between the authentication plugin and the LDAP server occurs according to this authentication method.

**Note**

For all simple LDAP authentication methods, it is recommended to also set TLS parameters to require that communication with the LDAP server take place over secure connections.

These authentication method values are permitted:

- **SIMPLE**: Use simple LDAP authentication. This method uses either one or two LDAP bind operations, depending on whether the MySQL account names an LDAP user distinguished name. See the description of `authentication_ldap_simple_bind_root_dn`.

- **AD-FOREST**: A variation on **SIMPLE**, such that authentication searches all domains in the Active Directory forest, performing an LDAP bind to each Active Directory domain until the user is found in some domain.

- `authentication_ldap_simple_bind_base_dn`

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AD-FOREST</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Command-Line Format</strong></td>
<td><code>--authentication-ldap-simple-bind-base-dn=value</code></td>
</tr>
<tr>
<td><strong>Introduced</strong></td>
<td>8.0.11</td>
</tr>
<tr>
<td><strong>System Variable</strong></td>
<td><code>authentication_ldap_simple_bind_base_dn</code></td>
</tr>
<tr>
<td><strong>Scope</strong></td>
<td>Global</td>
</tr>
<tr>
<td><strong>Dynamic</strong></td>
<td>Yes</td>
</tr>
<tr>
<td><strong>SET_VAR</strong> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>String</td>
</tr>
</tbody>
</table>
Pluggable Authentication System Variables

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>NULL</td>
</tr>
</tbody>
</table>

For simple LDAP authentication, the base distinguished name (DN). This variable can be used to limit the scope of searches by anchoring them at a certain location (the “base”) within the search tree.

Suppose that members of one set of LDAP user entries each have this form:

```plaintext
uid=user_name,ou=People,dc=example,dc=com
```

And that members of another set of LDAP user entries each have this form:

```plaintext
uid=user_name,ou=Admin,dc=example,dc=com
```

Then searches work like this for different base DN values:

- If the base DN is `ou=People,dc=example,dc=com`: Searches find user entries only in the first set.
- If the base DN is `ou=Admin,dc=example,dc=com`: Searches find user entries only in the second set.
- If the base DN is `ou=dc=example,dc=com`: Searches find user entries in the first or second set.

In general, more specific base DN values result in faster searches because they limit the search scope more.

**authentication_ldap_simple_bind_root_dn**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td>--authentication-ldap-simple-bind-root-dn=value</td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td>authentication_ldap_simple_bind_root_dn</td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td>NULL</td>
</tr>
</tbody>
</table>

For simple LDAP authentication, the root distinguished name (DN). This variable is used in conjunction with `authentication_ldap_simple_bind_root_pwd` as the credentials for authenticating to the LDAP server for the purpose of performing searches. Authentication uses either one or two LDAP bind operations, depending on whether the MySQL account names an LDAP user DN:

- If the account does not name a user DN: `authentication_ldap_simple` performs an initial LDAP binding using `authentication_ldap_simple_bind_root_dn` and `authentication_ldap_simple_bind_root_pwd`. (These are both empty by default, so if they are not set, the LDAP server must permit anonymous connections.) The resulting bind LDAP handle is used to search for the user DN, based on the client user name. `authentication_ldap_simple` performs a second bind using the user DN and client-supplied password.

- If the account does name a user DN: The first bind operation is unnecessary in this case. `authentication_ldap_simple` performs a single bind using the user DN and client-supplied password. This is faster than if the MySQL account does not specify an LDAP user DN.
### authentication_ldap_simple_bind_root_pwd

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--authentication-ldap-simple-bind-root-pwd=value</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>authentication_ldap_simple_bind_root_pwd</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><code>SET_VAR</code> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td>NULL</td>
</tr>
</tbody>
</table>

For simple LDAP authentication, the password for the root distinguished name. This variable is used in conjunction with `authentication_ldap_simple_bind_root_dn`. See the description of that variable.

### authentication_ldap_simple_ca_path

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--authentication-ldap-simple-ca-path=value</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>authentication_ldap_simple_ca_path</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><code>SET_VAR</code> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td>NULL</td>
</tr>
</tbody>
</table>

For simple LDAP authentication, the absolute path of the certificate authority file. Specify this file if it is desired that the authentication plugin perform verification of the LDAP server certificate.

**Note**

In addition to setting the `authentication_ldap_simple_ca_path` variable to the file name, you must add the appropriate certificate authority certificates to the file and enable the `authentication_ldap_simple_tls` system variable.

### authentication_ldap_simple_group_search_attr

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--authentication-ldap-simple-group-search-attr=value</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>authentication_ldap_simple_group_search_attr</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><code>SET_VAR</code> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
</tbody>
</table>
### Pluggable Authentication System Variables

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default Value</strong></td>
<td><code>cn</code></td>
</tr>
</tbody>
</table>

For simple LDAP authentication, the name of the attribute that specifies group names in LDAP directory entries. If `authentication_ldap_simple_group_search_attr` has its default value of `cn`, searches return the `cn` value as the group name. For example, if an LDAP entry with a `uid` value of `user1` has a `cn` attribute of `mygroup`, searches for `user1` return `mygroup` as the group name.

If the group search attribute is `isMemberOf`, LDAP authentication directly retrieves the user attribute `isMemberOf` value and assigns it as group information. If the group search attribute is not `isMemberOf`, LDAP authentication searches for all groups where the user is a member. (The latter is the default behavior.) This behavior is based on how LDAP group information can be stored two ways: 1) A group entry can have an attribute named `memberUid` or `member` with a value that is a user name; 2) A user entry can have an attribute named `isMemberOf` with values that are group names.

- **authentication_ldap_simple_group_search_filter**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Command-Line Format</strong></td>
<td><code>--authentication-ldap-simple-group-search-filter=value</code></td>
</tr>
<tr>
<td><strong>Introduced</strong></td>
<td>8.0.11</td>
</tr>
<tr>
<td><strong>System Variable</strong></td>
<td><code>authentication_ldap_simple_group_search_filter</code></td>
</tr>
<tr>
<td><strong>Scope</strong></td>
<td>Global</td>
</tr>
<tr>
<td><strong>Dynamic</strong></td>
<td>Yes</td>
</tr>
<tr>
<td><strong>SET_VAR Hint Applies</strong></td>
<td>No</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>String</td>
</tr>
</tbody>
</table>
| **Default Value** | `(|(&(objectClass=posixGroup)(memberUid=%s))
    (&(objectClass=group)(member=%s)))` |

For simple LDAP authentication, the custom group search filter.

The search filter value can contain `{UA}` and `{UD}` notation to represent the user name and the full user DN. For example, `{UA}` is replaced with a user name such as "admin", whereas `{UD}` is replaced with a use full DN such as "uid=admin,ou=People,dc=example,dc=com". The following value is the default, which supports both OpenLDAP and Active Directory:

```
(|{(objectClass=posixGroup)(memberUid=%s)})
  &{(objectClass=group)(member=%s)})
```

In some cases for the user scenario, `memberOf` is a simple user attribute that holds no group information. For additional flexibility, an optional `{GA}` prefix can be used with the group search attribute. Any group attribute with a `{GA}` prefix is treated as a user attribute having group names. For example, with a value of `{GA}MemberOf", if the group value is the DN, the first attribute value from the group DN is returned as the group name.

- **authentication_ldap_simple_init_pool_size**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Command-Line Format</strong></td>
<td><code>--authentication-ldap-simple-init-pool-size=#</code></td>
</tr>
<tr>
<td><strong>Introduced</strong></td>
<td>8.0.11</td>
</tr>
<tr>
<td><strong>System Variable</strong></td>
<td><code>authentication_ldap_simple_init_pool_size</code></td>
</tr>
<tr>
<td><strong>Scope</strong></td>
<td>Global</td>
</tr>
</tbody>
</table>
Pluggable Authentication System Variables

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>SET_VAR</strong> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>10</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>32767</td>
</tr>
</tbody>
</table>

For simple LDAP authentication, the initial size of the pool of connections to the LDAP server. Choose the value for this variable based on the average number of concurrent authentication requests to the LDAP server.

The plugin uses `authentication_ldap_simple_init_pool_size` and `authentication_ldap_simple_max_pool_size` together for connection-pool management:

- When the authentication plugin initializes, it creates `authentication_ldap_simple_init_pool_size` connections, unless `authentication_ldap_simple_max_pool_size=0` to disable pooling.

- If the plugin receives an authentication request when there are no free connections in the current connection pool, the plugin can create a new connection, up to the maximum connection pool size given by `authentication_ldap_simple_max_pool_size`.

- If the plugin receives a request when the pool size is already at its maximum and there are no free connections, authentication fails.

- When the plugin unloads, it closes all pooled connections.

Changes to plugin system variable settings may have no effect on connections already in the pool. For example, modifying the LDAP server host, port, or TLS settings does not affect existing connections. However, if the original variable values were invalid and the connection pool could not be initialized, the plugin attempts to reinitialize the pool for the next LDAP request. In this case, the new system variable values are used for the reinitialization attempt.

If `authentication_ldap_simple_max_pool_size=0` to disable pooling, each LDAP connection opened by the plugin uses the values the system variables have at that time.

- `authentication_ldap_simple_log_status`

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--authentication-ldap-simple-log-status=#</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>authentication_ldap_simple_log_status</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>SET_VAR</strong> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>1</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>1</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>(≥ 8.0.18)</td>
</tr>
</tbody>
</table>
For simple LDAP authentication, the logging level for messages written to the error log. The following table shows the permitted level values and their meanings.

<table>
<thead>
<tr>
<th>Option Value</th>
<th>Types of Messages Logged</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>No messages</td>
</tr>
<tr>
<td>2</td>
<td>Error messages</td>
</tr>
<tr>
<td>3</td>
<td>Error and warning messages</td>
</tr>
<tr>
<td>4</td>
<td>Error, warning, and information messages</td>
</tr>
<tr>
<td>5</td>
<td>Same as previous level plus debugging messages from MySQL</td>
</tr>
<tr>
<td>6</td>
<td>Same as previous level plus debugging messages from LDAP library</td>
</tr>
</tbody>
</table>

Log level 6 is available as of MySQL 8.0.18.

- **authentication_ldap_simple_max_pool_size**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--authentication-ldap-simple-max-pool-size=#</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>authentication_ldap_simple_max_pool_size</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><code>SET_VAR</code> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>1000</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>32767</td>
</tr>
</tbody>
</table>

For simple LDAP authentication, the maximum size of the pool of connections to the LDAP server. To disable connection pooling, set this variable to 0.

This variable is used in conjunction with `authentication_ldap_simple_init_pool_size`. See the description of that variable.

- **authentication_ldap_simple_referral**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td>`--authentication-ldap-simple-referral [=</td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.20</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>authentication_ldap_simple_referral</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><code>SET_VAR</code> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
</tbody>
</table>
Pluggable Authentication System Variables

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>OFF</td>
</tr>
</tbody>
</table>

For simple LDAP authentication, whether to enable LDAP search referral. See LDAP Search Referral.

- **authentication_ldap_simple_server_host**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--authentication-ldap-simple-server-host=host_name</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>authentication_ldap_simple_server_host</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
</tbody>
</table>

For simple LDAP authentication, the LDAP server host. The permitted values for this variable depend on the authentication method:

- For `authentication_ldap_simple_auth_method_name=SIMPLE`: The LDAP server host can be a host name or IP address.

- For `authentication_ldap_simple_auth_method_name=AD-FOREST`: The LDAP server host can be an Active Directory domain name. For example, for an LDAP server URL of `ldap://example.mem.local:389`, the server name can be `mem.local`.

An Active Directory forest setup can have multiple domains (LDAP server IPs), which can be discovered using DNS. On Unix and Unix-like systems, some additional setup may be required to configure your DNS server with SRV records that specify the LDAP servers for the Active Directory domain. Suppose that your configuration has these properties:

- The name server that provides information about Active Directory domains has IP address 10.172.166.100.

- The LDAP servers have names `ldap1.mem.local` through `ldap3.mem.local` and IP addresses 10.172.166.101 through 10.172.166.103.

You want the LDAP servers to be discoverable using SRV searches. For example, at the command line, a command like this should list the LDAP servers:

```
host -t SRV _ldap._tcp.mem.local
```

Perform the DNS configuration as follows:

1. Add a line to `/etc/resolv.conf` to specify the name server that provides information about Active Directory domains:

   ```
   nameserver 10.172.166.100
   ```

2. Configure the appropriate zone file for the name server with SRV records for the LDAP servers:

   ```
   _ldap._tcp.mem.local. 86400 IN SRV 0 100 389 ldap1.mem.local.
   _ldap._tcp.mem.local. 86400 IN SRV 0 100 389 ldap2.mem.local.
   ```
Pluggable Authentication System Variables

3. It may also be necessary to specify the IP address for the LDAP servers in `/etc/hosts` if the server host cannot be resolved. For example, add lines like this to the file:

```
10.172.166.101 ldap1.mem.local
10.172.166.102 ldap2.mem.local
10.172.166.103 ldap3.mem.local
```

With the DNS configured as just described, the server-side LDAP plugin can discover the LDAP servers and will try to authenticate in all domains until authentication succeeds or there are no more servers.

Windows needs no such settings as just described. Given the LDAP server host in the `authentication_ldap_simple_server_host` value, the Windows LDAP library searches all domains and attempts to authenticate.

- `authentication_ldap_simple_server_port`

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--authentication-ldap-simple-server-port=port_num</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>authentication_ldap_simple_server_port</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>389</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>1</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>32376</td>
</tr>
</tbody>
</table>

For simple LDAP authentication, the LDAP server TCP/IP port number.

As of MySQL 8.0.14, if the LDAP port number is configured as 636 or 3269, the plugin uses LDAPS (LDAP over SSL) instead of LDAP. (LDAPS differs from `startTLS`.)

- `authentication_ldap_simple_tls`

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td>`--authentication-ldap-simple-tls={OFF</td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>authentication_ldap_simple_tls</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
</tbody>
</table>
### The Connection-Control Plugins

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>OFF</td>
</tr>
</tbody>
</table>

For simple LDAP authentication, whether connections by the plugin to the LDAP server are secure. If this variable is enabled, the plugin uses TLS to connect securely to the LDAP server. If you enable this variable, you may also wish to set the `authentication_ldap_simple_ca_path` variable.

MySQL LDAP plugins support the StartTLS method, which initializes TLS on top of a plain LDAP connection. The `ldaps` method is deprecated and MySQL does not support it.

- `authentication_ldap_simple_user_search_attr`

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--authentication-ldap-simple-user-search-attr=value</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>authentication_ldap_simple_user_search_attr</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><code>SET_VAR</code> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td><code>uid</code></td>
</tr>
</tbody>
</table>

For simple LDAP authentication, the name of the attribute that specifies user names in LDAP directory entries. If a user distinguished name is not provided, the authentication plugin searches for the name using this attribute. For example, if the `authentication_ldap_simple_user_search_attr` value is `uid`, a search for the user name `user1` finds entries with a `uid` value of `user1`.

### 6.2 The Connection-Control Plugins

MySQL Server includes a plugin library that enables administrators to introduce an increasing delay in server response to clients after a certain number of consecutive failed connection attempts. This capability provides a deterrent that slows down brute force attacks that attempt to access MySQL user accounts. The plugin library contains two plugins:

- **CONNECTION_CONTROL** checks incoming connections and adds a delay to server responses as necessary. This plugin also exposes system variables that enable its operation to be configured and a status variable that provides rudimentary monitoring information.

  The `CONNECTION_CONTROL` plugin uses the audit plugin interface (see Writing Audit Plugins). To collect information, it subscribes to the `MYSQL_AUDIT_CONNECTION_CLASSMASK` event class, and processes `MYSQL_AUDIT_CONNECTION_CONNECT` and `MYSQL_AUDIT_CONNECTION_CHANGE_USER` subevents to check whether the server should introduce a delay before responding to client connection attempts.

- **CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS** implements an INFORMATION_SCHEMA table that exposes more detailed monitoring information for failed connection attempts.

The following sections provide information about connection-control plugin installation and configuration. For information about the `CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` table, see The INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS Table.

### 6.2.1 Connection-Control Plugin Installation
Connection-Control Plugin Installation

This section describes how to install the connection-control plugins, `CONNECTION_CONTROL` and `CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS`. For general information about installing plugins, see Installing and Uninstalling Plugins.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

The plugin library file base name is `connection_control`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

To load the plugins at server startup, use the `--plugin-load-add` option to name the library file that contains them. With this plugin-loading method, the option must be given each time the server starts. For example, put these lines in the server `my.cnf` file (adjust the `.so` suffix for your platform as necessary):

```
[mysqld]
plugin-load-add=connection_control.so
```

After modifying `my.cnf`, restart the server to cause the new settings to take effect.

Alternatively, to load the plugins at runtime, use these statements (adjust the `.so` suffix for your platform as necessary):

```
INSTALL PLUGIN CONNECTION_CONTROL
  SONAME 'connection_control.so';
INSTALL PLUGIN CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS
  SONAME 'connection_control.so';
```

`INSTALL PLUGIN` loads the plugin immediately, and also registers it in the `mysql.plugins` system table to cause the server to load it for each subsequent normal startup without the need for `--plugin-load-add`.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement (see Obtaining Server Plugin Information). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
FROM INFORMATION_SCHEMA.PLUGINS
WHERE PLUGIN_NAME LIKE 'connection%';
```

If a plugin fails to initialize, check the server error log for diagnostic messages.

If the plugins have been previously registered with `INSTALL PLUGIN` or are loaded with `--plugin-load-add`, you can use the `--connection-control` and `--connection-control-failed-login-attempts` options at server startup to control plugin activation. For example, to load the plugins at startup and prevent them from being removed at runtime, use these options:

```
[mysqld]
plugin-load-add=connection_control.so
connection-control=FORCE_PLUS_PERMANENT
connection-control-failed-login-attempts=FORCE_PLUS_PERMANENT
```

If it is desired to prevent the server from running without a given connection-control plugin, use an option value of `FORCE` or `FORCE_PLUS_PERMANENT` to force server startup to fail if the plugin does not initialize successfully.

**Note**

It is possible to install one plugin without the other, but both must be installed for full connection-control capability. In particular, installing only the
Connection-Control Plugin Installation

CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS plugin is of little use because without the CONNECTION_CONTROL plugin to provide the data that populates the CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS table, retrievals from the table will always be empty.

- Connection Delay Configuration
- Connection Failure Assessment
- Connection Failure Monitoring

Connection Delay Configuration

To enable you to configure its operation, the CONNECTION_CONTROL plugin exposes several system variables:

- connection_control_failed_connections_threshold: The number of consecutive failed connection attempts permitted to clients before the server adds a delay for subsequent connection attempts.

- connection_control_min_connection_delay: The amount of delay to add for each consecutive connection failure above the threshold.

- connection_control_max_connection_delay: The maximum delay to add.

To entirely disable checking for failed connection attempts, set connection_control_failed_connections_threshold to zero. If connection_control_failed_connections_threshold is nonzero, the amount of delay is zero up through that many consecutive failed connection attempts. Thereafter, the amount of delay is the number of failed attempts above the threshold, multiplied by connection_control_min_connection_delay milliseconds. For example, with the default connection_control_failed_connections_threshold and connection_control_min_connection_delay values of 3 and 1000, respectively, there is no delay for the first three consecutive failed connection attempts by a client, a delay of 1000 milliseconds for the fourth failed attempt, 2000 milliseconds for the fifth failed attempt, and so on, up to the maximum delay permitted by connection_control_max_connection_delay.

You can set the CONNECTION_CONTROL system variables at server startup or runtime. Suppose that you want to permit four consecutive failed connection attempts before the server starts delaying its responses, and to increase the delay by 1500 milliseconds for each additional failure after that. To set the relevant variables at server startup, put these lines in the server my.cnf file:

```sql
[mysqld]
plugin-load-add=connection_control.so
connection_control_failed_connections_threshold=4
connection_control_min_connection_delay=1500
```

To set and persist the variables at runtime, use these statements:

```sql
SET PERSIST connection_control_failed_connections_threshold = 4;
SET PERSIST connection_control_min_connection_delay = 1500;
```

SET PERSIST sets the value for the running MySQL instance. It also saves the value, causing it to be used for subsequent server restarts. To change a value for the running MySQL instance without saving it for subsequent restarts, use the GLOBAL keyword rather than PERSIST. See SET Syntax for Variable Assignment.

The connection_control_min_connection_delay and connection_control_max_connection_delay system variables have fixed minimum and maximum values of 1000 and 2147483647, respectively. In addition, the permitted range of values of each variable also depends on the current value of the other:

- connection_control_min_connection_delay cannot be set greater than the current value of connection_control_max_connection_delay.
• `connection_control_max_connection_delay` cannot be set less than the current value of `connection_control_min_connection_delay`.

Thus, to make the changes required for some configurations, you might need to set the variables in a specific order. Suppose that the current minimum and maximum delays are 1000 and 2000, and that you want to set them to 3000 and 5000. You cannot first set `connection_control_min_connection_delay` to 3000 because that is greater than the current `connection_control_max_connection_delay` value of 2000. Instead, set `connection_control_max_connection_delay` to 5000, then set `connection_control_min_connection_delay` to 3000.

**Connection Failure Assessment**

When the `CONNECTION_CONTROL` plugin is installed, it checks connection attempts and tracks whether they fail or succeed. For this purpose, a failed connection attempt is one for which the client user and host match a known MySQL account but the provided credentials are incorrect, or do not match any known account.

Failed-connection counting is based on the user/host combination for each connection attempt. Determination of the applicable user name and host name takes proxying into account and occurs as follows:

• If the client user proxies another user, the proxying user's information is used. For example, if `external_user@example.com` proxies `proxy_user@example.com`, connection counting uses the proxying user, `external_user@example.com`, rather than the proxied user, `proxy_user@example.com`. Both `external_user@example.com` and `proxy_user@example.com` must have valid entries in the `mysql.user` system table and a proxy relationship between them must be defined in the `mysql.proxies_priv` system table (see Section 4.18, “Proxy Users”).

• If the client user does not proxy another user, but does match a `mysql.user` entry, counting uses the `CURRENT_USER()` value corresponding to that entry. For example, if a user `user1` connecting from a host `host1.example.com` matches a `user1@host1.example.com` entry, counting uses `user1@host1.example.com`. If the user matches a `user1@%.example.com`, `user1@%.com`, or `user1@%` entry instead, counting uses `user1@%.example.com`, `user1@%.com`, or `user1@%`, respectively.

For the cases just described, the connection attempt matches some `mysql.user` entry, and whether the request succeeds or fails depends on whether the client provides the correct authentication credentials. For example, if the client presents an incorrect password, the connection attempt fails.

If the connection attempt matches no `mysql.user` entry, the attempt fails. In this case, no `CURRENT_USER()` value is available and connection-failure counting uses the user name provided by the client and the client host as determined by the server. For example, if a client attempts to connect as user `user2` from host `host2.example.com`, the user name part is available in the client request and the server determines the host information. The user/host combination used for counting is `user2@host2.example.com`.

**Note**

The server maintains information about which client hosts can possibly connect to the server (essentially the union of host values for `mysql.user` entries). If a client attempts to connect from any other host, the server rejects the attempt at an early stage of connection setup:

```
ERROR 1130 (HY000): Host 'host_name' is not allowed to connect to this MySQL server
```

Because this type of rejection occurs so early, `CONNECTION_CONTROL` does not see it, and does not count it.
Connection Failure Monitoring

To monitor failed connections, use these information sources:

- The `Connection_control_delay_generated` status variable indicates the number of times the server added a delay to its response to a failed connection attempt. This does not count attempts that occur before reaching the threshold defined by the `connection_control_failed_connections_threshold` system variable.

- The `INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` table provides information about the current number of consecutive failed connection attempts per client user/host combination. This counts all failed attempts, regardless of whether they were delayed.

Assigning a value to `connection_control_failed_connections_threshold` at runtime resets all accumulated failed-connection counters to zero, which has these visible effects:

- The `Connection_control_delay_generated` status variable is reset to zero.
- The `CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` table becomes empty.

6.2.2 Connection-Control System and Status Variables

This section describes the system and status variables that the `CONNECTION_CONTROL` plugin provides to enable its operation to be configured and monitored.

- Connection-Control System Variables
- Connection-Control Status Variables

Connection-Control System Variables

If the `CONNECTION_CONTROL` plugin is installed, it exposes these system variables:

- `connection_control_failed_connections_threshold`

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--connection-control-failed-connections-threshold=#</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.1</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>connection_control_failed_connections_threshold</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><code>SET_VAR</code> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>3</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>2147483647</td>
</tr>
</tbody>
</table>

The number of consecutive failed connection attempts permitted to clients before the server adds a delay for subsequent connection attempts:

- If the variable has a nonzero value \( N \), the server adds a delay beginning with consecutive failed attempt \( N+1 \). If a client has reached the point where connection responses are delayed, the delay also occurs for the next subsequent successful connection.

- Setting this variable to zero disables failed-connection counting. In this case, the server never adds delays.
For information about how `connection_control_failed_connections_threshold` interacts with other connection-control system and status variables, see Section 6.2.1, "Connection-Control Plugin Installation".

- **connection_control_max_connection_delay**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--connection-control-max-connection-delay=#</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.1</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>connection_control_max_connection_delay</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><code>SET_VAR</code> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>2147483647</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>1000</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>2147483647</td>
</tr>
</tbody>
</table>

The maximum delay in milliseconds for server response to failed connection attempts, if `connection_control_failed_connections_threshold` is greater than zero.

For information about how `connection_control_max_connection_delay` interacts with other connection-control system and status variables, see Section 6.2.1, "Connection-Control Plugin Installation".

- **connection_control_min_connection_delay**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--connection-control-min-connection-delay=#</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.1</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>connection_control_min_connection_delay</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><code>SET_VAR</code> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>1000</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>1000</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>2147483647</td>
</tr>
</tbody>
</table>

The minimum delay in milliseconds for server response to failed connection attempts, if `connection_control_failed_connections_threshold` is greater than zero. This is also the amount by which the server increases the delay for additional successive failures once it begins delaying.

For information about how `connection_control_min_connection_delay` interacts with other connection-control system and status variables, see Section 6.2.1, "Connection-Control Plugin Installation".

**Connection-Control Status Variables**

If the `CONNECTION_CONTROL` plugin is installed, it exposes this status variable:
The Password Validation Component

- **Connection_control_delay_generated**

  The number of times the server added a delay to its response to a failed connection attempt. This does not count attempts that occur before reaching the threshold defined by the `connection_control_failed_connections_threshold` system variable.

  This variable provides a simple counter. For more detailed connection-control monitoring information, examine the `INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` table; see The `INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` Table.

  Assigning a value to `connection_control_failed_connections_threshold` at runtime resets `Connection_control_delay_generated` to zero.

6.3 The Password Validation Component

The `validate_password` component serves to improve security by requiring account passwords and enabling strength testing of potential passwords. This component exposes system variables that enable you to configure password policy, and status variables for component monitoring.

- **Note**

  In MySQL 8.0, the `validate_password` plugin was reimplemented as the `validate_password` component. (For general information about server components, see MySQL Server Components.) The following instructions describe how to use the component, not the plugin. For instructions on using the plugin form of `validate_password`, see The Password Validation Plugin in MySQL 5.7 Reference Manual.

  The plugin form of `validate_password` is still available but is deprecated and will be removed in a future version of MySQL. MySQL installations that use the plugin should make the transition to using the component instead. See Section 6.3.3, “Transitioning to the Password Validation Component”.

The `validate_password` component implements these capabilities:

- For SQL statements that assign a password supplied as a cleartext value, `validate_password` checks the password against the current password policy and rejects the password if it is weak (the statement returns an `ER_NOT_VALID_PASSWORD` error). This applies to the `ALTER USER`, `CREATE USER`, and `SET PASSWORD` statements.

- For `CREATE USER` statements, `validate_password` requires that a password be given, and that it satisfies the password policy. This is true even if an account is locked initially because otherwise unlocking the account later would cause it to become accessible without a password that satisfies the policy.

- `validate_password` implements a `VALIDATE_PASSWORD_STRENGTH()` SQL function that assesses the strength of potential passwords. This function takes a password argument and returns an integer from 0 (weak) to 100 (strong).

- **Note**

  For statements that assign or modify account passwords (`ALTER USER`, `CREATE USER`, and `SET PASSWORD`), the `validate_password` capabilities described here apply only to accounts that use an authentication plugin that stores credentials internally to MySQL. For accounts that use plugins that perform authentication against a credentials system external to MySQL, password management must be handled externally against that system as well. For more information about internal credentials storage, see Section 4.15, “Password Management”.

216
The Password Validation Component

The preceding restriction does not apply to use of the `VALIDATE_PASSWORD_STRENGTH()` function because it does not affect accounts directly.

Examples:

- `validate_password` checks the cleartext password in the following statement. Under the default password policy, which requires passwords to be at least 8 characters long, the password is weak and the statement produces an error:

  ```
  mysql> ALTER USER USER() IDENTIFIED BY 'abc';
  ERROR 1819 (HY000): Your password does not satisfy the current policy requirements
  ```

- Passwords specified as hashed values are not checked because the original password value is not available for checking:

  ```
  mysql> ALTER USER 'jeffrey'@'localhost' IDENTIFIED WITH mysql_native_password AS '*0D3CED9BEC10A7777AE2C3CC353AAC084A633045E';
  Query OK, 0 rows affected (0.01 sec)
  ```

- This account-creation statement fails, even though the account is locked initially, because it does not include a password that satisfies the current password policy:

  ```
  mysql> CREATE USER 'juanita'@'localhost' ACCOUNT LOCK;
  ERROR 1819 (HY000): Your password does not satisfy the current policy requirements
  ```

- To check a password, use the `VALIDATE_PASSWORD_STRENGTH()` function:

  ```
  mysql> SELECT VALIDATE_PASSWORD_STRENGTH('weak');
  +------------------------------------+
  | VALIDATE_PASSWORD_STRENGTH('weak') |
  +------------------------------------+
  |                                 25 |
  +------------------------------------+
  mysql> SELECT VALIDATE_PASSWORD_STRENGTH('lessweak$_@123');
  +----------------------------------------------+
  | VALIDATE_PASSWORD_STRENGTH('lessweak$_@123') |
  +----------------------------------------------+
  |                                           50 |
  +----------------------------------------------+
  mysql> SELECT VALIDATE_PASSWORD_STRENGTH('N0Tweak$_@123!');
  +----------------------------------------------+
  | VALIDATE_PASSWORD_STRENGTH('N0Tweak$_@123!') |
  +----------------------------------------------+
  |                                          100 |
  +----------------------------------------------+
  ```

To configure password checking, modify the system variables having names of the form `validate_password.xxx`; these are the parameters that control password policy. See Section 6.3.2, “Password Validation Options and Variables”.

If `validate_password` is not installed, the `validate_password.xxx` system variables are not available, passwords in statements are not checked, and the `VALIDATE_PASSWORD_STRENGTH()` function always returns 0. For example, without the plugin installed, accounts can be assigned passwords shorter than 8 characters, or no password at all.

Assuming that `validate_password` is installed, it implements three levels of password checking: LOW, MEDIUM, and STRONG. The default is MEDIUM; to change this, modify the value of `validate_password.policy`. The policies implement increasingly strict password tests. The following descriptions refer to default parameter values, which can be modified by changing the appropriate system variables.
• **LOW** policy tests password length only. Passwords must be at least 8 characters long. To change this length, modify `validate_password.length`.

• **MEDIUM** policy adds the conditions that passwords must contain at least 1 numeric character, 1 lowercase character, 1 uppercase character, and 1 special (nonalphanumeric) character. To change these values, modify `validate_password.number_count`, `validate_password.mixed_case_count`, and `validate_password.special_char_count`.

• **STRONG** policy adds the condition that password substrings of length 4 or longer must not match words in the dictionary file, if one has been specified. To specify the dictionary file, modify `validate_password.dictionary_file`.

In addition, `validate_password` supports the capability of rejecting passwords that match the user name part of the effective user account for the current session, either forward or in reverse. To provide control over this capability, `validate_password` exposes a `validate_password.check_user_name` system variable, which is enabled by default.

### 6.3.1 Password Validation Component Installation and Uninstallation

This section describes how to install and uninstall the `validate_password` password-validation component. For general information about installing and uninstalling components, see MySQL Server Components.

**Note**

If you install MySQL 8.0 using the MySQL Yum repository, MySQL SLES Repository, or RPM packages provided by Oracle, the `validate_password` component is enabled by default after you start your MySQL Server for the first time.

Upgrades to MySQL 8.0 from 5.7 using Yum or RPM packages leave the `validate_password` plugin in place. To make the transition from the `validate_password` plugin to the `validate_password` component, see Section 6.3.3, “Transitioning to the Password Validation Component”.

To be usable by the server, the component library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

To install the `validate_password` component, use this statement:

```
INSTALL COMPONENT 'file://component_validate_password';
```

Component installation is a one-time operation that need not be done per server startup. `INSTALL COMPONENT` loads the component, and also registers it in the `mysql.component` system table to cause it to be loaded during subsequent server startups.

To uninstall the `validate_password` component, use this statement:

```
UNINSTALL COMPONENT 'file://component_validate_password';
```

`UNINSTALL COMPONENT` unloads the component, and deregisters it from the `mysql.component` system table to cause it not to be loaded during subsequent server startups.

### 6.3.2 Password Validation Options and Variables

This section describes the system and status variables that `validate_password` provides to enable its operation to be configured and monitored.
Password Validation Options and Variables

- Password Validation Component System Variables
- Password Validation Component Status Variables
- Password Validation Plugin Options
- Password Validation Plugin System Variables
- Password Validation Plugin Status Variables

Password Validation Component System Variables

If the `validate_password` component is enabled, it exposes several system variables that enable configuration of password checking:

```
mysql> SHOW VARIABLES LIKE 'validate_password.%';
+--------------------------------------+--------+
| Variable_name                        | Value  |
+--------------------------------------+--------+
| validate_password.check_user_name    | ON     |
| validate_password.dictionary_file    |        |
| validate_password.length             | 8      |
| validate_password.mixed_case_count   | 1      |
| validate_password.number_count       | 1      |
| validate_password.policy             | MEDIUM |
| validate_password.special_char_count | 1      |
+--------------------------------------+--------+
```

To change how passwords are checked, you can set these system variables at server startup or at runtime. The following list describes the meaning of each variable.

- `validate_password.check_user_name`
  
<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td>`--validate-password.check-user-name=[OFF</td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.4</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>validate_password.check_user_name</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>ON</td>
</tr>
</tbody>
</table>

Whether `validate_password` compares passwords to the user name part of the effective user account for the current session and rejects them if they match. This variable is unavailable unless `validate_password` is installed.

By default, `validate_password.check_user_name` is enabled. This variable controls user name matching independent of the value of `validate_password.policy`.

When `validate_password.check_user_name` is enabled, it has these effects:

- Checking occurs in all contexts for which `validate_password` is invoked, which includes use of statements such as `ALTER USER` or `SET PASSWORD` to change the current user’s password, and invocation of functions such as `VALIDATE_PASSWORD_STRENGTH()`.

- The user names used for comparison are taken from the values of the `USER()` and `CURRENT_USER()` functions for the current session. An implication is that a user who has sufficient privileges to set another user’s password can set the password to that user’s name, and cannot set that user’s password to the name of the user executing the statement. For example,
'root'@'localhost' can set the password for 'jeffrey'@'localhost' to 'jeffrey', but cannot set the password to 'root'.

- Only the user name part of the USER() and CURRENT_USER() function values is used, not the host name part. If a user name is empty, no comparison occurs.
- If a password is the same as the user name or its reverse, a match occurs and the password is rejected.
- User-name matching is case sensitive. The password and user name values are compared as binary strings on a byte-by-byte basis.
- If a password matches the user name, VALIDATE_PASSWORD_STRENGTH() returns 0 regardless of how other validate_password system variables are set.

- **validate_password.dictionary_file**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td>--validate-password.dictionary-file=file_name</td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.4</td>
</tr>
<tr>
<td>System Variable</td>
<td>validate_password.dictionary_file</td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>File name</td>
</tr>
</tbody>
</table>

The path name of the dictionary file that validate_password uses for checking passwords. This variable is unavailable unless validate_password is installed.

By default, this variable has an empty value and dictionary checks are not performed. For dictionary checks to occur, the variable value must be nonempty. If the file is named as a relative path, it is interpreted relative to the server data directory. File contents should be lowercase, one word per line. Contents are treated as having a character set of utf8. The maximum permitted file size is 1MB.

For the dictionary file to be used during password checking, the password policy must be set to 2 (STRONG); see the description of the validate_password.policy system variable. Assuming that is true, each substring of the password of length 4 up to 100 is compared to the words in the dictionary file. Any match causes the password to be rejected. Comparisons are not case sensitive.

For VALIDATE_PASSWORD_STRENGTH(), the password is checked against all policies, including STRONG, so the strength assessment includes the dictionary check regardless of the validate_password.policy value.

validate_password.dictionary_file can be set at runtime and assigning a value causes the named file to be read without a server restart.

- **validate_password.length**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td>--validate-password.length=#</td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.4</td>
</tr>
<tr>
<td>System Variable</td>
<td>validate_password.length</td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
</tbody>
</table>
Password Validation Options and Variables

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>8</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
</tbody>
</table>

The minimum number of characters that `validate_password` requires passwords to have. This variable is unavailable unless `validate_password` is installed.

The `validate_password.length` minimum value is a function of several other related system variables. The value cannot be set less than the value of this expression:

\[
\text{validate_password.number_count} + \text{validate_password.special_char_count} + (2 \times \text{validate_password.mixed_case_count})
\]

If `validate_password` adjusts the value of `validate_password.length` due to the preceding constraint, it writes a message to the error log.

- `validate_password.mixed_case_count`

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--validate-password.mixed-case-count=#</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.4</td>
</tr>
<tr>
<td>System Variable</td>
<td>validate_password.mixed_case_count</td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>1</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
</tbody>
</table>

The minimum number of lowercase and uppercase characters that `validate_password` requires passwords to have if the password policy is MEDIUM or stronger. This variable is unavailable unless `validate_password` is installed.

For a given `validate_password.mixed_case_count` value, the password must have that many lowercase characters, and that many uppercase characters.

- `validate_password.number_count`

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--validate-password.number-count=#</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.4</td>
</tr>
<tr>
<td>System Variable</td>
<td>validate_password.number_count</td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>1</td>
</tr>
</tbody>
</table>
Password Validation Options and Variables

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
</tbody>
</table>

The minimum number of numeric (digit) characters that `validate_password` requires passwords to have if the password policy is `MEDIUM` or stronger. This variable is unavailable unless `validate_password` is installed.

- `validate_password.policy`

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--validate-password.policy=value</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.4</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>validate_password.policy</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><code>SET_VAR</code> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Default Value</td>
<td>1</td>
</tr>
<tr>
<td>Valid Values</td>
<td>0, 1, 2</td>
</tr>
</tbody>
</table>

The password policy enforced by `validate_password`. This variable is unavailable unless `validate_password` is installed.

`validate_password.policy` affects how `validate_password` uses its other policy-setting system variables, except for checking passwords against user names, which is controlled independently by `validate_password.check_user_name`.

The `validate_password.policy` value can be specified using numeric values 0, 1, 2, or the corresponding symbolic values `LOW`, `MEDIUM`, `STRONG`. The following table describes the tests performed for each policy. For the length test, the required length is the value of the `validate_password.length` system variable. Similarly, the required values for the other tests are given by other `validate_password.xxx` variables.

<table>
<thead>
<tr>
<th>Policy</th>
<th>Tests Performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 or <code>LOW</code></td>
<td>Length</td>
</tr>
<tr>
<td>1 or <code>MEDIUM</code></td>
<td>Length; numeric, lowercase/uppercase, and special characters</td>
</tr>
<tr>
<td>2 or <code>STRONG</code></td>
<td>Length; numeric, lowercase/uppercase, and special characters; dictionary file</td>
</tr>
</tbody>
</table>

- `validate_password.special_char_count`

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--validate-password.special-char-count=#</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.4</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>validate_password.special_char_count</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><code>SET_VAR</code> Hint Applies</td>
<td>No</td>
</tr>
</tbody>
</table>
Password Validation Options and Variables

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>1</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
</tbody>
</table>

The minimum number of nonalphanumeric characters that `validate_password` requires passwords to have if the password policy is MEDIUM or stronger. This variable is unavailable unless `validate_password` is installed.

Password Validation Component Status Variables

If the `validate_password` component is enabled, it exposes status variables that provide operational information:

```
mysql> SHOW STATUS LIKE 'validate_password.%';
+-----------------------------------------------+---------------------+
| Variable_name                                 | Value               |
+-----------------------------------------------+---------------------+
| validate_password.dictionary_file_last_parsed | 2019-10-03 08:33:49 |
| validate_password.dictionary_file_words_count | 1902                |
+-----------------------------------------------+---------------------+
```

The following list describes the meaning of each status variable.

- `validate_password.dictionary_file_last_parsed`
  When the dictionary file was last parsed. This variable is unavailable unless `validate_password` is installed.

- `validate_password.dictionary_file_words_count`
  The number of words read from the dictionary file. This variable is unavailable unless `validate_password` is installed.

Password Validation Plugin Options

**Note**

In MySQL 8.0, the `validate_password` plugin was reimplemented as the `validate_password` component. The `validate_password` plugin is deprecated and will be removed in a future version of MySQL. Consequently, its options are also deprecated and will be removed. MySQL installations that use the plugin should make the transition to using the component instead. See Section 6.3.3, “Transitioning to the Password Validation Component”.

To control activation of the `validate_password` plugin, use this option:

- `--validate-password[=value]`

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--validate-password[=value]</code></td>
</tr>
<tr>
<td>Type</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Default Value</td>
<td>ON</td>
</tr>
<tr>
<td>Valid Values</td>
<td>ON, OFF, FORCE, FORCE_PLUS_PERMANENT</td>
</tr>
</tbody>
</table>
This option controls how the server loads the deprecated `validate_password` plugin at startup. The value should be one of those available for plugin-loading options, as described in Installing and Uninstalling Plugins. For example, `--validate-password=FORCE_PLUS_PERMANENT` tells the server to load the plugin at startup and prevents it from being removed while the server is running.

This option is available only if the `validate_password` plugin has been previously registered with `INSTALL PLUGIN` or is loaded with `--plugin-load-add`. See Section 6.3.1, "Password Validation Component Installation and Uninstallation".

### Password Validation Plugin System Variables

**Note**

In MySQL 8.0, the `validate_password` plugin was reimplemented as the `validate_password` component. The `validate_password` plugin is deprecated and will be removed in a future version of MySQL. Consequently, its system variables are also deprecated and will be removed. Use the corresponding system variables of the `validate_password` component; see Password Validation Component System Variables. MySQL installations that use the plugin should make the transition to using the component instead. See Section 6.3.3, "Transitioning to the Password Validation Component".

- **validate_password_check_user_name**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td>`--validate-password-check-user-name=[{OFF</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>validate_password_check_user_name</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>SET_VAR</strong> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>ON</td>
</tr>
</tbody>
</table>

This `validate_password` plugin system variable is deprecated and will be removed in a future version of MySQL. Use the corresponding `validate_password.check_user_name` system variable of the `validate_password` component instead.

- **validate_password_dictionary_file**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--validate-password-dictionary-file=file_name</code></td>
</tr>
<tr>
<td>System Variable</td>
<td><code>validate_password_dictionary_file</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>SET_VAR</strong> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>File name</td>
</tr>
</tbody>
</table>

This `validate_password` plugin system variable is deprecated and will be removed in a future version of MySQL. Use the corresponding `validate_password.dictionary_file` system variable of the `validate_password` component instead.

- **validate_password_length**
### Password Validation Options and Variables

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--validate-password-length=#</code></td>
</tr>
<tr>
<td>System Variable</td>
<td><code>validate_password_length</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>8</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
</tbody>
</table>

This `validate_password` plugin system variable is deprecated and will be removed in a future version of MySQL. Use the corresponding `validate_password.length` system variable of the `validate_password` component instead.

- `validate_password.mixed_case_count`

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--validate-password-mixed-case-count=#</code></td>
</tr>
<tr>
<td>System Variable</td>
<td><code>validate_password_mixed_case_count</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>1</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
</tbody>
</table>

This `validate_password` plugin system variable is deprecated and will be removed in a future version of MySQL. Use the corresponding `validate_password.mixed_case_count` system variable of the `validate_password` component instead.

- `validate_password.number_count`

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--validate-password-number-count=#</code></td>
</tr>
<tr>
<td>System Variable</td>
<td><code>validate_password_number_count</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>1</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
</tbody>
</table>

This `validate_password` plugin system variable is deprecated and will be removed in a future version of MySQL. Use the corresponding `validate_password.number_count` system variable of the `validate_password` component instead.
Password Validation Options and Variables

- **validate_password_policy**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--validate-password-policy=value</code></td>
</tr>
<tr>
<td>System Variable</td>
<td><code>validate_password_policy</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Default Value</td>
<td>1</td>
</tr>
<tr>
<td>Valid Values</td>
<td>0, 1, 2</td>
</tr>
</tbody>
</table>

This `validate_password` plugin system variable is deprecated and will be removed in a future version of MySQL. Use the corresponding `validate_password.policy` system variable of the `validate_password` component instead.

- **validate_password_special_char_count**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--validate-password-special-char-count=#</code></td>
</tr>
<tr>
<td>System Variable</td>
<td><code>validate_password_special_char_count</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>1</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
</tbody>
</table>

This `validate_password` plugin system variable is deprecated and will be removed in a future version of MySQL. Use the corresponding `validate_password.special_char_count` system variable of the `validate_password` component instead.

### Password Validation Plugin Status Variables

**Note**

In MySQL 8.0, the `validate_password` plugin was reimplemented as the `validate_password` component. The `validate_password` plugin is deprecated and will be removed in a future version of MySQL. Consequently, its status variables are also deprecated and will be removed. Use the corresponding status variables of the `validate_password` component; see Password Validation Component Status Variables. MySQL installations that use the plugin should make the transition to using the component instead. See Section 6.3.3, "Transitioning to the Password Validation Component".

- **validate_password_dictionary_file_last_parsed**

This `validate_password` plugin status variable is deprecated and will be removed in a future version of MySQL. Use the corresponding
validate_password.dictionary_file_last_parsed status variable of the validate_password component instead.

• validate_password_dictionary_file_words_count

This validate_password plugin status variable is deprecated and will be removed in a future version of MySQL. Use the corresponding validate_password.dictionary_file_words_count status variable of the validate_password component instead.

6.3.3 Transitioning to the Password Validation Component

Note

In MySQL 8.0, the validate_password plugin was reimplemented as the validate_password component. The validate_password plugin is deprecated and will be removed in a future version of MySQL.

MySQL installations that currently use the validate_password plugin should make the transition to using the validate_password component instead. To do so, use the following procedure. The procedure installs the component before uninstalling the plugin, to avoid having a time window during which no password validation occurs. (The component and plugin can be installed simultaneously. In this case, the server attempts to use the component, falling back to the plugin if the component is unavailable.)

1. Install the validate_password component:

   INSTALL COMPONENT 'file://component_validate_password';

2. Test the validate_password component to ensure that it works as expected. If you need to set any validate_password.xxx system variables, you can do so at runtime using SET GLOBAL. (Any option file changes that must be made are performed in the next step.)

3. Adjust any references to the plugin system and status variables to refer to the corresponding component system and status variables. Suppose that previously you had configured the plugin at startup using an option file like this:

   [mysqld]
   validate-password=FORCE_PLUS_PERMANENT
   validate_password_dictionary_file=/usr/share/dict/words
   validate_password_length=10
   validate_password_number_count=2

   Those settings are appropriate for the plugin, but must be modified to apply to the component. To adjust the option file, omit the --validate-password option (it applies only to the plugin, not the component), and modify the system variable references from no-dot names appropriate for the plugin to dotted names appropriate for the component:

   [mysqld]
   validate_password.dictionary_file=/usr/share/dict/words
   validate_password.length=10
   validate_password.number_count=2

   Similar adjustments are needed for applications that refer at runtime to validate_password plugin system and status variables. Change the no-dot plugin variable names to the corresponding dotted component variable names.

4. Uninstall the validate_password plugin:

   UNINSTALL PLUGIN validate_password;

   If the validate_password plugin is loaded at server startup using a --plugin-load or --plugin-load-add option, omit that option from the server startup procedure. For example, if the option is listed in a server option file, remove it from the file.
5. Restart the server.

### 6.4 The MySQL Keyring

MySQL Server supports a keyring that enables internal server components and plugins to securely store sensitive information for later retrieval. The implementation is plugin-based:

- The `keyring_file` plugin stores keyring data in a file local to the server host. This plugin is available in all MySQL distributions, Community Edition and Enterprise Edition included. See Section 6.4.2, "Using the keyring_file File-Based Plugin".

- The `keyring_encrypted_file` plugin stores keyring data in an encrypted file local to the server host. This plugin is available in MySQL Enterprise Edition distributions. See Section 6.4.3, "Using the keyring_encrypted_file Keyring Plugin".

- `keyring_okv` is a KMIP 1.1 plugin for use with KMIP-compatible back end keyring storage products such as Oracle Key Vault and Gemalto SafeNet KeySecure Appliance. This plugin is available in MySQL Enterprise Edition distributions. See Section 6.4.4, "Using the keyring_okv KMIP Plugin".

- The `keyring_aws` plugin communicates with the Amazon Web Services Key Management Service for key generation and uses a local file for key storage. This plugin is available in MySQL Enterprise Edition distributions. See Section 6.4.5, "Using the keyring_aws Amazon Web Services Keyring Plugin".

- MySQL 8.0.18 and higher includes `keyring_hashicorp`, a plugin that communicates with HashiCorp Vault for back end storage. This plugin is available in MySQL Enterprise Edition distributions. See Section 6.4.6, "Using the HashiCorp Vault Keyring Plugin".

- A MySQL server operational mode enables migration of keys between underlying keyring keystores. This enables DBAs to switch a MySQL installation from one keyring plugin to another. See Section 6.4.7, "Migrating Keys Between Keyring Keystores".

- An SQL interface for keyring key management is implemented as a set of user-defined functions (UDFs). See Section 6.4.9, "General-Purpose Keyring Key-Management Functions".

- In MySQL 8.0.16 and higher, the `keyring_keys` table exposes metadata for keys in the keyring. Key metadata includes key IDs, key owners, and backend key IDs. The `keyring_keys` table does not expose any sensitive keyring data such as key contents. See The keyring_keys table.

#### Warning

The `keyring_file` and `keyring_encrypted_file` plugins for encryption key management are not intended as a regulatory compliance solution. Security standards such as PCI, FIPS, and others require use of key management systems to secure, manage, and protect encryption keys in key vaults or hardware security modules (HSMs).

Uses for the keyring within MySQL include:

- The InnoDB storage engine uses the keyring to store its key for tablespace encryption. InnoDB can use any supported keyring plugin.

- MySQL Enterprise Audit uses the keyring to store the audit log file encryption password. The audit log plugin can use any supported keyring plugin.

- When binary log encryption has been activated for a MySQL server (by setting `binlog_encryption=ON`), the binary log encryption keys used to encrypt the file passwords for the binary log files and relay log files are stored in the keyring. Any supported keyring plugin can be used to store binary log encryption keys. Binary log encryption keys are retained as long as there are files on the server that were encrypted using them. When the binary log master key is rotated manually, all binary log encryption keys that no longer apply to any retained binary log files or relay log files are cleared from the keyring. If a retained binary log file or relay log file cannot be initialized for re-
encryption, the relevant binary log encryption keys are not deleted in case the files can be recovered in the future. For example, this might be the case if a file listed in a binary log index file is currently unreadable, or if a channel fails to initialize. For more information, see Encrypting Binary Log Files and Relay Log Files.

For general keyring installation instructions, see Section 6.4.1, “Keyring Plugin Installation”. For information specific to a given keyring plugin, see the section describing that plugin.

For information about using the keyring UDFs, see Section 6.4.9, “General-Purpose Keyring Key-Management Functions”.

Keyring plugins and UDFs access a keyring service that provides the interface for server components to the keyring. For information about accessing the keyring plugin service and writing keyring plugins, see The Keyring Service, and Writing Keyring Plugins.

### 6.4.1 Keyring Plugin Installation

Keyring service consumers require a keyring plugin to be installed. MySQL provides these plugin choices:

- **keyring_file**: A plugin that stores keyring data in a file local to the server host. Available in all MySQL distributions.

- **keyring_encrypted_file**: A plugin that stores keyring data in an encrypted file local to the server host. Available in MySQL Enterprise Edition distributions.

- **keyring_okv**: A plugin that uses KMIP-compatible back end keyring storage products such as Oracle Key Vault and Gemalto SafeNet KeySecure Appliance. Available in MySQL Enterprise Edition distributions.

- **keyring_aws**: A plugin that communicates with the Amazon Web Services Key Management Service as a back end for key generation and uses a local file for key storage. Available in MySQL Enterprise Edition distributions.

- **keyring_hashicorp**: A plugin that communicates with HashiCorp Vault for back end storage. Available in MySQL Enterprise Edition distributions.

This section describes how to install the keyring plugin of your choosing. For general information about installing plugins, see Installing and Uninstalling Plugins.

If you intend to use keyring user-defined functions (UDFs) in conjunction with the keyring plugin, install the UDFs following keyring installation using the instructions in Section 6.4.9, “General-Purpose Keyring Key-Management Functions”.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

Installation for each keyring plugin is similar. The following instructions use `keyring_file`. Users of a different keyring plugin can substitute its name for `keyring_file`.

The `keyring_file` plugin library file base name is `keyring_file`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

#### Note

Only one keyring plugin should be enabled at a time. Enabling multiple keyring plugins is unsupported and results may not be as anticipated.

The keyring plugin must be loaded early during the server startup sequence so that server components can access it as necessary during their own initialization. For example, the InnoDB storage engine uses the keyring for tablespace encryption, so the keyring plugin must be loaded and available prior to InnoDB initialization.
To load the plugin, use the `--early-plugin-load` option to name the plugin library file that contains it. For example, on platforms where the plugin library file suffix is `.so`, use these lines in the server `my.cnf` file (adjust the `.so` suffix for your platform as necessary):

```ini
[mysqld]
early-plugin-load=keyring_file.so
```

Before starting the server, check the notes for your chosen keyring plugin to see whether it permits or requires additional configuration:

- For `keyring_file`: Section 6.4.2, “Using the keyring_file File-Based Plugin”.
- For `keyring_okv`: Section 6.4.4, “Using the keyring_okv KMIP Plugin”.
- For `keyring_aws`: Section 6.4.5, “Using the keyring_aws Amazon Web Services Keyring Plugin”
- For `keyring_hashicorp`: Section 6.4.6, “Using the HashiCorp Vault Keyring Plugin”

After performing any plugin-specific configuration, verify plugin installation. With the MySQL server running, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement (see Obtaining Server Plugin Information). For example:

```sql
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS FROM INFORMATION_SCHEMA.PLUGINS WHERE PLUGIN_NAME LIKE 'keyring%';
+--------------+---------------+
| PLUGIN_NAME  | PLUGIN_STATUS |
+--------------+---------------+
| keyring_file | ACTIVE        |
+--------------+---------------+
```

If the plugin fails to initialize, check the server error log for diagnostic messages.

If no keyring plugin is available when a server component tries to access the keyring service, the service cannot be used by that component. As a result, the component may fail to initialize or may initialize with limited functionality. For example, if InnoDB finds that there are encrypted tablespaces when it initializes, it attempts to access the keyring. If the keyring is unavailable, InnoDB can access only unencrypted tablespaces. To ensure that InnoDB can access encrypted tablespaces as well, use `--early-plugin-load` to load the keyring plugin.

Plugins can be loaded by other methods, such as the `--plugin-load` or `--plugin-load-add` option or the `INSTALL PLUGIN` statement. However, keyring plugins loaded using those methods may be available too late in the server startup sequence for certain server components, such as InnoDB:

- Plugin loading using `--plugin-load` or `--plugin-load-add` occurs after InnoDB initialization.
- Plugins installed using `INSTALL PLUGIN` are registered in the `mysql.plugin` system table and loaded automatically for subsequent server restarts. However, because `mysql.plugin` is an InnoDB table, any plugins named in it can be loaded during startup only after InnoDB initialization.

### 6.4.2 Using the keyring_file File-Based Plugin

The `keyring_file` plugin is a keyring plugin that stores keyring data in a file local to the server host.

**Warning**

The `keyring_file` plugin for encryption key management is not intended as a regulatory compliance solution. Security standards such as PCI, FIPS, and others require use of key management systems to secure, manage, and protect encryption keys in key vaults or hardware security modules (HSMs).

To install the `keyring_file` plugin, use the general keyring installation instructions found in Section 6.4.1, “Keyring Plugin Installation”, together with the configuration information specific to `keyring_file` found here.
To be usable during the server startup process, `keyring_file` must be loaded using the `--early-plugin-load` option. The `keyring_file_data` system variable optionally configures the location of the file used by the `keyring_file` plugin for data storage. The default value is platform specific. To configure the file location explicitly, set the variable value at startup. For example, use these lines in the server `my.cnf` file (adjust the `.so` suffix and file location for your platform as necessary):

```
[mysqld]
early-plugin-load=keyring_file.so
keyring_file_data=/usr/local/mysql/mysql-keyring/keyring
```

Keyring operations are transactional: The `keyring_file` plugin uses a backup file during write operations to ensure that it can roll back to the original file if an operation fails. The backup file has the same name as the value of the `keyring_file_data` system variable with a suffix of `.backup`.

For additional information about `keyring_file_data`, see Section 6.4.12, “Keyring System Variables”.

To ensure that keys are flushed only when the correct keyring storage file exists, `keyring_file` stores a SHA-256 checksum of the keyring in the file. Before updating the file, the plugin verifies that it contains the expected checksum.

The `keyring_file` plugin supports the functions that comprise the standard MySQL Keyring service interface. Keyring operations performed by those functions are accessible at two levels:

- SQL interface: In SQL statements, call the user-defined functions (UDFs) described in Section 6.4.9, “General-Purpose Keyring Key-Management Functions”.
- C interface: In C-language code, call the keyring service functions described in The Keyring Service.

Example (using UDFs):

```
SELECT keyring_key_generate('MyKey', 'AES', 32);
SELECT keyring_key_remove('MyKey');
```

For information about the key types permitted by `keyring_file`, see Section 6.4.8, “Supported Keyring Key Types and Lengths”.

### 6.4.3 Using the keyring_encrypted_file Keyring Plugin

**Note**

The `keyring_encrypted_file` plugin is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see https://www.mysql.com/products/.

The `keyring_encrypted_file` plugin is a keyring plugin that stores keyring data in an encrypted file local to the server host.

**Warning**

The `keyring_encrypted_file` plugin for encryption key management is not intended as a regulatory compliance solution. Security standards such as PCI, FIPS, and others require use of key management systems to secure, manage, and protect encryption keys in key vaults or hardware security modules (HSMs).

To install the `keyring_encrypted_file` plugin, use the general keyring installation instructions found in Section 6.4.1, “Keyring Plugin Installation”, together with the configuration information specific to `keyring_encrypted_file` found here.

To be usable during the server startup process, `keyring_encrypted_file` must be loaded using the `--early-plugin-load` option. To specify the password for encrypting the keyring data file, set the `keyring_encrypted_file_password` system variable. (The password is
Using the keyring_okv KMIP Plugin

mandatory; if not specified at server startup, keyring_encrypted_file initialization fails.) The keyring_encrypted_file_data system variable optionally configures the location of the file used by the keyring_encrypted_file plugin for data storage. The default value is platform specific. To configure the file location explicitly, set the variable value at startup. For example, use these lines in the server my.cnf file (adjust the .so suffix and file location for your platform as necessary and substitute your chosen password):

```
[mysqld]
early-plugin-load=keyring_encrypted_file.so
keyring_encrypted_file_data=/usr/local/mysql/mysql-keyring/keyring-encrypted
keyring_encrypted_file_password=password
```

Because the my.cnf file stores a password when written as shown, it should have a restrictive mode and be accessible only to the account used to run the MySQL server.

Keyring operations are transactional: The keyring_encrypted_file plugin uses a backup file during write operations to ensure that it can roll back to the original file if an operation fails. The backup file has the same name as the value of the keyring_encrypted_file_data system variable with a suffix of .backup.

For additional information about the system variables used to configure the keyring_encrypted_file plugin, see Section 6.4.12, "Keyring System Variables".

To ensure that keys are flushed only when the correct keyring storage file exists, keyring_encrypted_file stores a SHA-256 checksum of the keyring in the file. Before updating the file, the plugin verifies that it contains the expected checksum. In addition, keyring_encrypted_file encrypts file contents using AES before writing the file, and decrypts file contents after reading the file.

The keyring_encrypted_file plugin supports the functions that comprise the standard MySQL Keyring service interface. Keyring operations performed by those functions are accessible at two levels:

- SQL interface: In SQL statements, call the user-defined functions (UDFs) described in Section 6.4.9, "General-Purpose Keyring Key-Management Functions".
- C interface: In C-language code, call the keyring service functions described in The Keyring Service.

Example (using UDFs):

```
SELECT keyring_key_generate('MyKey', 'AES', 32);
SELECT keyring_key_remove('MyKey');
```

For information about the key types permitted by keyring_encrypted_file, see Section 6.4.8, "Supported Keyring Key Types and Lengths".

### 6.4.4 Using the keyring_okv KMIP Plugin

Note

The keyring_okv plugin is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see https://www.mysql.com/products/.

The Key Management Interoperability Protocol (KMIP) enables communication of cryptographic keys between a key management server and its clients. The keyring_okv keyring plugin uses the KMIP 1.1 protocol to communicate securely as a client of a KMIP back end. Keyring material is generated exclusively by the back end, not by keyring_okv. The plugin works with these KMIP-compatible products:

- Oracle Key Vault
- Gemalto SafeNet KeySecure Appliance
The `keyring_okv` plugin supports the functions that comprise the standard MySQL Keyring service interface. Keyring operations performed by those functions are accessible at two levels:

- **SQL interface:** In SQL statements, call the user-defined functions (UDFs) described in Section 6.4.9, “General-Purpose Keyring Key-Management Functions”.
- **C interface:** In C-language code, call the keyring service functions described in The Keyring Service.

Example (using UDFs):

```sql
SELECT keyring_key_generate('MyKey', 'AES', 32);
SELECT keyring_key_remove('MyKey');
```

For information about the key types permitted by `keyring_okv`, Section 6.4.8, “Supported Keyring Key Types and Lengths”.

To install the `keyring_okv` plugin, use the general keyring installation instructions found in Section 6.4.1, “Keyring Plugin Installation”, together with the configuration information specific to `keyring_okv` found here.

- General `keyring_okv` Configuration
- Configuring `keyring_okv` for Oracle Key Vault
- Configuring `keyring_okv` for Gemalto SafeNet KeySecure Appliance
- Password-Protecting the `keyring_okv` Key File

**General `keyring_okv` Configuration**

Regardless of which KMIP back end the `keyring_okv` plugin uses for keyring storage, the `keyring_okv_conf_dir` system variable configures the location of the directory used by `keyring_okv` for its support files. The default value is empty, so you must set the variable to name a properly configured directory before the plugin can communicate with the KMIP back end. Unless you do so, `keyring_okv` writes a message to the error log during server startup that it cannot communicate:

```
[Warning] Plugin keyring_okv reported: 'For keyring_okv to be initialized, please point the keyring_okv_conf_dir variable to a directory containing Oracle Key Vault configuration file and ssl materials'
```

The `keyring_okv_conf_dir` variable must name a directory that contains the following items:

- **okvclient.ora:** A file that contains details of the KMIP back end with which `keyring_okv` will communicate.
- **ssl:** A directory that contains the certificate and key files required to establish a secure connection with the KMIP back end: `CA.pem`, `cert.pem`, and `key.pem`. If the key file is password-protected, the `ssl` directory can contain a single-line text file named `password.txt` containing the password needed to decrypt the key file.

Both the `okvclient.ora` file and `ssl` directory with the certificate and key files are required for `keyring_okv` to work properly. The procedure used to populate the configuration directory with these files depends on the KMIP back end used with `keyring_okv`, as described elsewhere.

The configuration directory used by `keyring_okv` as the location for its support files should have a restrictive mode and be accessible only to the account used to run the MySQL server. For example, on Unix and Unix-like systems, to use the `/usr/local/mysql/mysql-keyring-okv` directory, the following commands (executed as `root`) create the directory and set its mode and ownership:

```
cd /usr/local/mysql
mkdir mysql-keyring-okv
chmod 750 mysql-keyring-okv
chown mysql mysql-keyring-okv
chgrp mysql mysql-keyring-okv
```
To be usable during the server startup process, `keyring_okv` must be loaded using the `--early-plugin-load` option. Also, set the `keyring_okv_conf_dir` system variable to tell `keyring_okv` where to find its configuration directory. For example, use these lines in the server `my.cnf` file (adjust the `.so` suffix and directory location for your platform as necessary):

```
[mysqld]
early-plugin-load=keyring_okv.so
keyring_okv_conf_dir=/usr/local/mysql/mysql-keyring-okv
```

For additional information about `keyring_okv_conf_dir`, see Section 6.4.12, “Keyring System Variables”.

Configuring `keyring_okv` for Oracle Key Vault

The discussion here assumes that you are familiar with Oracle Key Vault. Some pertinent information sources:

- Oracle Key Vault site
- Oracle Key Vault documentation

In Oracle Key Vault terminology, clients that use Oracle Key Vault to store and retrieve security objects are called endpoints. To communicate with Oracle Key Vault, it is necessary to register as an endpoint and enroll by downloading and installing endpoint support files.

The following procedure briefly summarizes the process of setting up `keyring_okv` for use with Oracle Key Vault:

1. Create the configuration directory for the `keyring_okv` plugin to use.
2. Register an endpoint with Oracle Key Vault to obtain an enrollment token.
3. Use the enrollment token to obtain the `okvclient.jar` client software download.
4. Install the client software to populate the `keyring_okv` configuration directory that contains the Oracle Key Vault support files.

Use the following procedure to configure `keyring_okv` and Oracle Key Vault to work together. This description only summarizes how to interact with Oracle Key Vault. For details, visit the Oracle Key Vault site and consult the Oracle Key Vault Administrator’s Guide.

1. Create the configuration directory that will contain the Oracle Key Vault support files, and make sure that the `keyring_okv_conf_dir` system variable is set to name that directory (for details, see General `keyring_okv` Configuration).
2. Log in to the Oracle Key Vault management console as a user who has the System Administrator role.
4. Provide the required endpoint information and click Register. The endpoint type should be Other. Successful registration results in an enrollment token.
5. Log out from the Oracle Key Vault server.
6. Connect again to the Oracle Key Vault server, this time without logging in. Use the endpoint enrollment token to enroll and request the `okvclient.jar` software download. Save this file to your system.
7. Install the `okvclient.jar` file using the following command (you must have JDK 1.4 or higher):

```
java -jar okvclient.jar -d dir_name [-v]
```

The directory name following the `-d` option is the location in which to install extracted files. The `-v` option, if given, causes log information to be produced that may be useful if the command fails.
Using the keyring_okv KMIP Plugin

When the command asks for an Oracle Key Vault endpoint password, do not provide one. Instead, press Enter. (The result is that no password will be required when the endpoint connects to Oracle Key Vault.)

8. The preceding command produces an okvclient.ora file, which should be in this location under the directory named by the -d option in the preceding java -jar command:

install_dir/conf/okvclient.ora

The file contents include lines that look something like this:

SERVER=host_ip:port_num
STANDBY_SERVER=host_ip:port_num

The keyring_okv plugin attempts to communicate with the server running on the host named by the SERVER variable and falls back to STANDBY_SERVER if that fails:

• For the SERVER variable, a setting in the okvclient.ora file is mandatory.
• For the STANDBY_SERVER variable, a setting in the okvclient.ora file is optional.

9. Go to the Oracle Key Vault installer directory and test the setup by running this command:

okvutil/bin/okvutil list

The output should look something like this:

Unique ID                               Type            Identifier
255AB8DE-C97F-482C-E053-0100007F28B9 Symmetric Key -
264BF6E0-A20E-7C42-E053-0100007FB29C Symmetric Key -

For a fresh Oracle Key Vault server (a server without any key in it), the output looks like this instead, to indicate that there are no keys in the vault:

no objects found

10. Use this command to extract the ssl directory containing SSL materials from the okvclient.jar file:

jar xf okvclient.jar ssl

11. Copy the Oracle Key Vault support files (the okvclient.ora file and the ssl directory) into the configuration directory.

12. (Optional) If you wish to password-protect the key file, use the instructions in Password-Protecting the keyring_okv Key File.

After completing the preceding procedure, restart the MySQL server. It loads the keyring_okv plugin and keyring_okv uses the files in its configuration directory to communicate with Oracle Key Vault.

Configuring keyring_okv for Gemalto SafeNet KeySecure Appliance

Gemalto SafeNet KeySecure Appliance uses the KMIP protocol (version 1.1 or 1.2). The keyring_okv keyring plugin (which supports KMIP 1.1) can use KeySecure as its KMIP back end for keyring storage.

Use the following procedure to configure keyring_okv and KeySecure to work together. The description only summarizes how to interact with KeySecure. For details, consult the section named Add a KMIP Server in the KeySecure User Guide.

1. Create the configuration directory that will contain the KeySecure support files, and make sure that the keyring_okv_conf_dir system variable is set to name that directory (for details, see General keyring_okv Configuration).
Using the keyring_okv KMIP Plugin

2. In the configuration directory, create a subdirectory named `ssl` to use for storing the required SSL certificate and key files.

3. In the configuration directory, create a file named `okvclient.ora`. It should have following format:

```
SERVER=host_ip:port_num
STANDBY_SERVER=host_ip:port_num
```

For example, if KeySecure is running on host 198.51.100.20 and listening on port 9002, the `okvclient.ora` file looks like this:

```
SERVER=198.51.100.20:9002
STANDBY_SERVER=198.51.100.20:9002
```

4. Connect to the KeySecure Management Console as an administrator with credentials for Certificate Authorities access.

5. Navigate to Security >> Local CAs and create a local certificate authority (CA).

6. Go to Trusted CA Lists. Select Default and click on Properties. Then select Edit for Trusted Certificate Authority List and add the CA just created.

7. Download the CA and save it in the `ssl` directory as a file named `CA.pem`.

8. Navigate to Security >> Certificate Requests and create a certificate. Then you will be able to download a compressed tar file containing certificate PEM files.

9. Extract the PEM files from in the downloaded file. For example, if the file name is `csr_w_pk_pkcs8.gz`, decompress and unpack it using this command:

```
tar zxvf csr_w_pk_pkcs8.gz
```

Two files result from the extraction operation: `certificate_request.pem` and `private_key_pkcs8.pem`.

10. Use this `openssl` command to decrypt the private key and create a file named `key.pem`:

```
openssl pkcs8 -in private_key_pkcs8.pem -out key.pem
```

11. Copy the `key.pem` file into the `ssl` directory.

12. Copy the certificate request in `certificate_request.pem` into the clipboard.

13. Navigate to Security >> Local CAs. Select the same CA that you created earlier (the one you downloaded to create the `CA.pem` file), and click Sign Request. Paste the Certificate Request from the clipboard, choose a certificate purpose of Client (the keyring is a client of KeySecure), and click Sign Request. The result is a certificate signed with the selected CA in a new page.

14. Copy the signed certificate to the clipboard, then save the clipboard contents as a file named `cert.pem` in the `ssl` directory.

15. (Optional) If you wish to password-protect the key file, use the instructions in Password-Protecting the keyring_okv Key File.

After completing the preceding procedure, restart the MySQL server. It loads the `keyring_okv` plugin and `keyring_okv` uses the files in its configuration directory to communicate with KeySecure.

Password-Protecting the keyring_okv Key File

You can optionally protect the key file with a password and supply a file containing the password to enable the key file to be decrypted. To do so, change location to the `ssl` directory and perform these steps:

1. Encrypt the `key.pem` key file. For example, use a command like this, and enter the encryption password at the prompts:
Using the keyring_aws Amazon Web Services Keyring Plugin

shell> openssl rsa -des3 -in key.pem -out key.pem.new
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:

2. Save the encryption password in a single-line text file named password.txt in the ssl directory.

3. Verify that the encrypted key file can be decrypted using the following command. The decrypted file should display on the console:

shell> openssl rsa -in key.pem.new -passin file:password.txt

4. Remove the original key.pem file and rename key.pem.new to key.pem.

5. Change the ownership and access mode of new key.pem file and password.txt file as necessary to ensure that they have the same restrictions as other files in the ssl directory.

6.4.5 Using the keyring_aws Amazon Web Services Keyring Plugin

Note

The keyring_aws plugin is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see https://www.mysql.com/products/.

The keyring_aws plugin is a keyring plugin that communicates with the Amazon Web Services Key Management Service (AWS KMS) as a back end for key generation and uses a local file for key storage. All keyring material is generated exclusively by the AWS server, not by keyring_aws.

keyring_aws is available on these platforms:

• Debian 8
• EL7
• macOS 10.13 and 10.14
• SLES 12
• Ubuntu 14.04 and 16.04
• Windows

The discussion here assumes that you are familiar with AWS in general and KMS in particular. Some pertinent information sources:

• AWS site
• KMS documentation

The following sections provide configuration and usage information for the keyring_aws keyring plugin:

• keyring_aws Configuration
• keyring_aws Operation
• keyring_aws Credential Changes

keyring_aws Configuration

To install the keyring_aws plugin, use the general keyring installation instructions found in Section 6.4.1, “Keyring Plugin Installation”, together with the plugin-specific configuration information found here.
The plugin library file contains the `keyring_aws` plugin and two user-defined functions (UDFs), `keyring_aws_rotate_cmk()` and `keyring_aws_rotate_keys()`.

To configure `keyring_aws`, you must obtain a secret access key that provides credentials for communicating with AWS KMS and write it to a configuration file:

1. Create an AWS KMS account.
2. Use AWS KMS to create a secret access key ID and secret access key. The access key serves to verify your identity and that of your applications.
3. Use the AWS KMS account to create a customer master key (CMK) ID. At MySQL startup, set the `keyring_aws_cmk_id` system variable to the CMK ID value. This variable is mandatory and there is no default. (Its value can be changed at runtime if desired using `SET GLOBAL`.)
4. If necessary, create the directory in which the configuration file will be located. The directory should have a restrictive mode and be accessible only to the account used to run the MySQL server. For example, on Unix and Unix-like systems, to use `/usr/local/mysql/mysql-keyring` as the file name, the following commands (executed as root) create its parent directory and set the directory mode and ownership:

```
shell> cd /usr/local/mysql
shell> mkdir mysql-keyring
shell> chmod 750 mysql-keyring
shell> chown mysql mysql-keyring
shell> chgrp mysql mysql-keyring
```

At MySQL startup, set the `keyring_aws_conf_file` system variable to `/usr/local/mysql/mysql-keyring/keyring_aws_conf` to indicate the configuration file location to the server.

5. Prepare the `keyring_aws` configuration file, which should contain two lines:
   - Line 1: The secret access key ID
   - Line 2: The secret access key

For example, if the key ID is `wwwwwwwwwwwwwEXAMPLE` and the key is `xxxxxxxxxxxxx/yyyyyyy/zzzzzzzzEXAMPLEKEY`, the configuration file looks like this:

```
wwwwwwwwwwwwwEXAMPLE
xxxxxxxxxxxxx/yyyyyyy/zzzzzzzzEXAMPLEKEY
```

To be usable during the server startup process, `keyring_aws` must be loaded using the `--early-plugin-load` option. The `keyring_aws_cmk_id` system variable is mandatory and configures the customer master key (CMK) ID obtained from the AWS KMS server. The `keyring_aws_conf_file` and `keyring_aws_data_file` system variables optionally configure the locations of the files used by the `keyring_aws` plugin for configuration information and data storage. The file location variable default values are platform specific. To configure the locations explicitly, set the variable values at startup. For example, use these lines in the server `my.cnf` file (adjust the `.so` suffix and file locations for your platform as necessary):

```
[mysqld]
early-plugin-load=keyring_aws.so
keyring_aws_cmk_id='arn:aws:kms:us-west-2:111122223333:key/abcd1234-ef56-ab12-cd34-ef56abcd1234'
keyring_aws_conf_file=/usr/local/mysql/mysql-keyring/keyring_aws_conf
keyring_aws_data_file=/usr/local/mysql/mysql-keyring/keyring_aws_data
```

For the `keyring_aws` plugin to start successfully, the configuration file must exist and contain valid secret access key information, initialized as described previously. The storage file need not exist. If it does not, `keyring_aws` attempts to create it (as well as its parent directory, if necessary).

For additional information about the system variables used to configure the `keyring_aws` plugin, see Section 6.4.12, “Keyring System Variables”.
Using the keyring_aws Amazon Web Services Keyring Plugin

Start the MySQL server and install the UDFs associated with the keyring_aws plugin. This is a one-time operation, performed by executing the following statements (adjust the .so suffix for your platform as necessary):

```sql
CREATE FUNCTION keyring_aws_rotate_cmk RETURNS INTEGER SONAME 'keyring_aws.so';
CREATE FUNCTION keyring_aws_rotate_keys RETURNS INTEGER SONAME 'keyring_aws.so';
```

For additional information about the keyring_aws UDFs, see Section 6.4.10, “Plugin-Specific Keyring Key-Management Functions”.

**keyring_aws Operation**

At plugin startup, the keyring_aws plugin reads the AWS secret access key ID and key from its configuration file. It also reads any encrypted keys contained in its storage file into its in-memory cache.

During operation, keyring_aws maintains encrypted keys in the in-memory cache and uses the storage file as local persistent storage. Each keyring operation is transactional: keyring_aws either successfully changes both the in-memory key cache and the keyring storage file, or the operation fails and the keyring state remains unchanged.

To ensure that keys are flushed only when the correct keyring storage file exists, keyring_aws stores a SHA-256 checksum of the keyring in the file. Before updating the file, the plugin verifies that it contains the expected checksum.

The keyring_aws plugin supports the functions that comprise the standard MySQL Keyring service interface. Keyring operations performed by these functions are accessible at two levels:

- C interface: In C-language code, call the keyring service functions described in The Keyring Service.
- SQL interface: In SQL statements, call the user-defined functions (UDFs) described in Section 6.4.9, “General-Purpose Keyring Key-Management Functions”.

Example (using UDFs):

```sql
SELECT keyring_key_generate('MyKey', 'AES', 32);
SELECT keyring_key_remove('MyKey');
```

In addition, the keyring_aws_rotate_cmk() and keyring_aws_rotate_keys() UDFs “extend” the keyring plugin interface to provide AWS-related capabilities not covered by the standard keyring service interface. These capabilities are accessible only by calling the UDFs. There are no corresponding C-language key service functions.

For information about the key types permitted by keyring_aws, see Section 6.4.8, “Supported Keyring Key Types and Lengths”.

**keyring_aws Credential Changes**

Assuming that the keyring_aws plugin has initialized properly at server startup, it is possible to change the credentials used for communicating with AWS KMS:

1. Use AWS KMS to create a new secret access key ID and secret access key.

2. Store the new credentials in the configuration file (the file named by the keyring_aws_conf_file system variable). The file format is as described previously.

3. Reinitialize the keyring_aws plugin so that it rereads the configuration file. Assuming that the new credentials are valid, the plugin should initialize successfully.

There are two ways to reinitialize the plugin:

- Restart the server. This is simpler and has no side effects, but is not suitable for installations that require minimal server downtime with as few restarts as possible.
Using the HashiCorp Vault Keyring Plugin

- Reinitialize the plugin without restarting the server by executing the following statements (adjust the .so suffix for your platform as necessary):

  UNINSTALL PLUGIN keyring_aws;
  INSTALL PLUGIN keyring_aws SONAME 'keyring_aws.so';

**Note**

In addition to loading a plugin at runtime, INSTALL PLUGIN has the side effect of registering the plugin in the mysql.plugin system table. Because of this, if you decide to stop using keyring_aws, it is not sufficient to remove the --early-plugin-load option from the set of options used to start the server. That stops the plugin from loading early, but the server still attempts to load it when it gets to the point in the startup sequence where it loads the plugins registered in mysql.plugin.

Consequently, if you execute the UNINSTALL PLUGIN plus INSTALL PLUGIN sequence just described to change the AWS KMS credentials, then to stop using keyring_aws, it is necessary to execute UNINSTALL PLUGIN again to unregister the plugin in addition to removing the --early-plugin-load option.

### 6.4.6 Using the HashiCorp Vault Keyring Plugin

**Note**

The keyring_hashicorp plugin is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see https://www.mysql.com/products/.

The keyring_hashicorp plugin is a keyring plugin that communicates with HashiCorp Vault for back end storage. The plugin supports HashiCorp Vault AppRole authentication. All keys are stored in HashiCorp Vault. No key information is permanently stored in MySQL server local storage. (An optional in-memory key cache may be used as intermediate storage.) Random key generation is performed on the MySQL server side, and the keys are subsequently stored to Hashicorp Vault.

The keyring_hashicorp plugin supports the functions that comprise the standard MySQL Keyring service interface. Keyring operations performed by those functions are accessible at two levels:

- **SQL interface**: In SQL statements, call the user-defined functions (UDFs) described in Section 6.4.9, “General-Purpose Keyring Key-Management Functions”.

- **C interface**: In C-language code, call the keyring service functions described in The Keyring Service.

Example (using UDFs):

```sql
SELECT keyring_key_generate('MyKey', 'AES', 32);
SELECT keyring_key_remove('MyKey');
```

For information about the key types permitted by keyring_hashicorp, see Section 6.4.8, “Supported Keyring Key Types and Lengths”.

To install the keyring_hashicorp plugin, use the general keyring installation instructions found in Section 6.4.1, “Keyring Plugin Installation”, together with the configuration information specific to keyring_hashicorp found here. Plugin-specific information includes preparation of the certificate and key files needed for connecting to HashiCorp Vault, as well as configuring Vault itself. The following sections provide the necessary instructions.

- **Certificate and Key Preparation**

- **HashiCorp Vault Setup**
Using the HashiCorp Vault Keyring Plugin

- keyring_hashicorp Configuration

**Certificate and Key Preparation**

The keyring_hashicorp plugin requires a secure connection to the HashiCorp Vault server, employing the HTTPS protocol. A typical setup includes a set of certificate and key files:

- **company.crt**: A custom CA certificate belonging to the organization. This file is used both by HashiCorp Vault server and the keyring_hashicorp plugin.
- **vault.key**: The private key of the HashiCorp Vault server instance. This file is used by HashiCorp Vault server.
- **vault.crt**: The certificate of the HashiCorp Vault server instance. This file must be signed by the organization CA certificate.

The following instructions describe how to create the certificate and key files using OpenSSL. (If you already have the files, proceed to HashiCorp Vault Setup.) The instructions as shown apply to Linux platforms and may require adjustment for other platforms.

**Important**

Certificates generated by these instructions are self-signed, which may not be very secure. After you gain experience using such files, consider obtaining certificate/key material from a registered certificate authority.

1. Prepare the company and HashiCorp Vault server keys.

Use the following commands to generate the key files:

```bash
openssl genrsa -aes256 -out company.key 4096
openssl genrsa -aes256 -out vault.key 2048
```

The commands produce files holding the company private key (company.key) and the Vault server private key (vault.key). The keys are randomly generated RSA keys of 4,096 and 2,048 bits, respectively.

Each command prompts for a password. (For testing purposes, the password is not required. To disable it, omit the -aes256 argument.)

The key files hold sensitive information and should be stored in a secure location. The password (also sensitive) is required later, so write it down and store it in a secure location.

(Optional) To check key file content and validity, use the following commands:

```bash
openssl rsa -in company.key -check
openssl rsa -in vault.key -check
```

2. Create the company CA certificate.

Use the following command to create a company CA certificate file named company.crt that is valid for 365 days (enter the command on a single line):

```bash
openssl req -x509 -new -nodes -key company.key -sha256 -days 365 -out company.crt
```

You will be prompted for the company key password during CA certificate creation, if you used the -aes256 argument to perform key encryption during key generation. You will also be prompted for information about the certificate holder (that is, you or your company), as shown here:

```
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
```
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:

Answer the prompts with appropriate values.

3. Create a certificate signing request.

To create a HashiCorp Vault server certificate, a Certificate Signing Request (CSR) must be prepared for the newly created server key. Create a configuration file named `request.conf` containing the following lines. If the HashiCorp Vault server does not run on the local host, substitute appropriate CN and IP values, and make any other changes required.

```
[req]
distinguished_name = vault
x509_extensions = v3_req
prompt = no
[vault]
C = US
ST = CA
L = RWC
O = Company
CN = 127.0.0.1
[v3_req]
subjectAltName = @alternatives
authorityKeyIdentifier = keyid,issuer
basicConstraints = CA:TRUE
[alternatives]
IP = 127.0.0.1
```

Use this command to create the signing request:

```
openssl req -new -key vault.key -config request.conf -out request.csr
```

The output file (`request.csr`) is an intermediate file that serves as input for creation of the server certificate.

4. Create the HashiCorp Vault server certificate.

Sign the combined information from the HashiCorp Vault server key (`vault.key`) and the CSR (`request.csr`) with the company certificate (`company.crt`) to create the HashiCorp Vault server certificate (`vault.crt`). Use the following command to do this (enter the command on a single line):

```
openssl x509 -req -in request.csr -CA company.crt -CAkey company.key -CAcreateserial -out vault.crt -days 365 -sha256
```

To make the `vault.crt` server certificate useful, append the contents of the `company.crt` company certificate to it. This is required so that the company certificate is delivered along with the server certificate in requests.

```
cat company.crt >> vault.crt
```

If you open the `vault.crt` file with a text editor, its content should look like this:

```
-----BEGIN CERTIFICATE-----
... content of HashiCorp Vault server certificate ...
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
... content of company certificate ...
-----END CERTIFICATE-----
```

HashiCorp Vault Setup

The following instructions describe how to create a HashiCorp Vault setup that facilitates testing the `keyring_hashicorp` plugin.
Using the HashiCorp Vault Keyring Plugin

Important

A test setup is similar to a production setup, but production use of HashiCorp Vault entails additional security considerations such as use of non-self-signed certificates and storing the company certificate in the system trust store. It is assumed that you will implement whatever additional security steps are needed to satisfy your operational requirements.

These instructions assume availability of the certificate and key files created in Certificate and Key Preparation. See that section if you do not have the files.

1. Fetch the HashiCorp Vault binary.

   Download the HashiCorp Vault binary appropriate for your platform from https://www.vaultproject.io/downloads.html.

   Extract the content of the archive to produce the executable vault command, which is used to perform HashiCorp Vault operations. If necessary, add the directory where you install the command to the system path.

   (Optional) HashiCorp Vault supports autocomplete options that make it easier to use. For more information, see https://learn.hashicorp.com/vault/getting-started/install#command-completion.

2. Create the HashiCorp Vault server configuration file.

   Prepare a configuration file named config.hcl with the following content. For the tls_cert_file, tls_key_file, and path values, substitute path names appropriate for your system.

   ```hcl
   listener "tcp" {
     address="127.0.0.1:8200"
     tls_cert_file="/home/username/certificates/vault.crt"
     tls_key_file="/home/username/certificates/vault.key"
   }
   storage "file" {
     path = "/home/username/vaultstorage/storage"
   }
   ui = true
   ```

3. Start the HashiCorp Vault server.

   To start the Vault server, use the following command, where the -config option specifies the path to the configuration file just created:

   ```bash
   vault server -config=config.hcl
   ```

   During this step, you may be prompted for a password for the Vault server private key stored in the vault.key file. The server should start, displaying some information on the console (IP, port, and so forth).

   So that you can enter the remaining commands, put the vault server command in the background or open another terminal before continuing.

4. Initialize the HashiCorp Vault server.

   Note

   The operations described in this step are required only when starting Vault the first time, to obtain the unseal key and root token. Subsequent Vault instance restarts require only unsealing using the unseal key.

   Issue the following commands (assuming Bourne shell syntax):

   ```bash
   vault init
   ```
Using the HashiCorp Vault Keyring Plugin

```bash
export VAULT_SKIP_VERIFY=1
vault operator init -n 1 -t 1
```

The first command enables the `vault` command to temporarily ignore the fact that no company certificate has been added to the system trust store. It compensates for the fact that our self-signed CA is not added to that store. (For production use, such a certificate should be added.)

The second command creates a single unseal key with a requirement for a single unseal key to be present for unsealing. (For production use, an instance would have multiple unseal keys with up to that many keys required to be entered to unseal it. The unseal keys should be delivered to key custodians within the company. Use of a single key might be considered a security issue because that permits the vault to be unsealed by a single key custodian.)

Vault should reply with information about the unseal key and root token, plus some additional text (the actual unseal key and root token values will differ from those shown here):

```bash
... Unseal Key 1: I2xwcFQc89200Nt2p8iRN1nkHtUzWS+JybL39BjOE= Initial Root Token: s.vTvXeo3tPEYehfcd9WH7oUKz ...
```

Store the unseal key and root token in a secure location.

5. **Unseal the HashiCorp Vault server.**

Use this command to unseal the Vault server:

```bash
vault operator unseal
```

When prompted to enter the unseal key, use the key obtained previously during Vault initialization. Vault should produce output indicating that setup is complete and the vault is unsealed.

6. **Log in to the HashiCorp Vault server and verify its status.**

Prepare the environment variables required for logging in as root:

```bash
vault login s.vTvXeo3tPEYehfcd9WH7oUKz
```

For the token value in that command, substitute the content of the root token obtained previously during Vault initialization.

Verify the Vault server status:

```bash
vault status
```

The output should contain these lines (among others):

```bash
... Initialized true Sealed false ...
```

7. **Set up HashiCorp Vault authentication and storage.**

**Note**

The operations described in this step are needed only the first time the Vault instance is run. They need not be repeated afterward.

Enable the AppRole authentication method and verify that it is in the authentication method list:

```bash
vault auth enable approle
```
Using the HashiCorp Vault Keyring Plugin

```bash
vault auth list
```

Enable the Vault KeyValue storage engine:

```bash
vault secrets enable -version=1 kv
```

Create and set up a role for use with the `keyring_hashicorp` plugin (enter the command on a single line):

```bash
vault write auth/approle/role/mysql token_num_uses=0
token_ttl=20m token_max_ttl=30m secret_id_num_uses=0
```

8. Add an AppRole security policy.

**Note**

The operations described in this step are needed only the first time the Vault instance is run. They need not be repeated afterward.

Prepare a policy that to permit the previously created role to access appropriate secrets. Create a new file named `mysql.hcl` with the following content:

```hcl
path "kv/mysql/*" {
    capabilities = ["create", "read", "update", "delete", "list"]
}
```

Import the policy file to the Vault server to create a policy named `mysql-policy`, then assign the policy to the new role:

```bash
vault policy write mysql-policy mysql.hcl
vault write auth/approle/role/mysql policies=mysql-policy
```

Obtain the ID of the newly created role and store it in a secure location:

```bash
vault read auth/approle/role/mysql/role-id
```

Generate a secret ID for the role and store it in a secure location:

```bash
vault write -f auth/approle/role/mysql/secret-id
```

After these AppRole role ID and secret ID credentials are generated, they are expected to remain valid indefinitely. They need not be generated again and the `keyring_hashicorp` plugin can be configured with them for use on an ongoing basis. For more information about AuthRole authentication, visit [https://www.vaultproject.io/docs/auth/approle.html](https://www.vaultproject.io/docs/auth/approle.html).

**keyring_hashicorp Configuration**

The plugin library file contains the `keyring_hashicorp` plugin and a user-defined function (UDF), `keyring_hashicorp_update_config()`. When the plugin initializes and terminates, it automatically loads and unloads the UDF, so there is no need to load and unload the UDF manually.

The `keyring_hashicorp` plugin supports the configuration arguments shown in the following table, specified by assigning values to a set of system variables.

<table>
<thead>
<tr>
<th>Configuration Parameter</th>
<th>System Variable</th>
<th>Mandatory</th>
</tr>
</thead>
<tbody>
<tr>
<td>HashiCorp Server URL</td>
<td><code>keyring_hashicorp_server_url</code></td>
<td></td>
</tr>
<tr>
<td>AppRole role ID</td>
<td><code>keyring_hashicorp_role_id</code></td>
<td>Yes</td>
</tr>
<tr>
<td>AppRole secret ID</td>
<td><code>keyring_hashicorp_secret_id</code></td>
<td>Yes</td>
</tr>
<tr>
<td>Store path</td>
<td><code>keyring_hashicorp_store_path</code></td>
<td>Yes</td>
</tr>
<tr>
<td>Authorization Path</td>
<td><code>keyring_hashicorp_auth_path</code></td>
<td></td>
</tr>
</tbody>
</table>
Using the HashiCorp Vault Keyring Plugin

<table>
<thead>
<tr>
<th>Configuration Parameter</th>
<th>System Variable</th>
<th>Mandatory</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA certificate file path</td>
<td>keyring_hashicorp_ca_path</td>
<td></td>
</tr>
<tr>
<td>Caching</td>
<td>keyring_hashicorp_caching</td>
<td></td>
</tr>
</tbody>
</table>

To be usable during the server startup process, `keyring_hashicorp` must be loaded using the `--early-plugin-load` option. As indicated by the preceding table, several plugin-related system variables are mandatory and must also be set. For example, use these lines in the server `my.cnf` file (adjust the `.so` suffix and file locations for your platform as necessary):

```
[mysqld]
early-plugin-load=keyring_hashicorp.so
keyring_hashicorp_role_id='ee3b495c-d0c9-11e9-8881-8444c71c32aa'
keyring_hashicorp_secret_id='0512af29-d0ca-11e9-95ee-0010e00dd718'
keyring_hashicorp_store_path=/v1/kv/mysql
```

MySQL Server authenticates against HashiCorp Vault using AppRole authentication. Successful authentication requires that two secrets be provided to Vault, a role ID and a secret ID, which are similar in concept to user name and password. The role ID and secret ID values to use are those obtained during the HashiCorp Vault setup procedure performed previously. To specify the two IDs, assign their respective values to the `keyring_hashicorp_role_id` and `keyring_hashicorp_secret_id` system variables. The setup procedure also results in a store path of `/v1/kv/mysql`, which is the value to assign to `keyring_hashicorp_commit_store_path`.

At plugin initialization time, `keyring_hashicorp` attempts to connect to the HashiCorp Vault server using the configuration values. If the connection is successful, the plugin stores the values in corresponding system variables that have `_commit_` in their name. For example, upon successful connection, the plugin stores the values of `keyring_hashicorp_role_id` and `keyring_hashicorp_store_path` in `keyring_hashicorp_commit_role_id` and `keyring_hashicorp_commit_store_path`.

Reconfiguration at runtime can be performed with the assistance of the `keyring_hashicorp_update_config()` UDF:

1. Use `SET` statements to assign the desired new values to the configuration system variables shown in the preceding table. These assignments in themselves have no effect on ongoing plugin operation.

2. Invoke `keyring_hashicorp_update_config()` to cause the plugin to reconfigure and reconnect to the HashiCorp Vault server using the new variable values.

3. If the connection is successful, the plugin stores the updated configuration values in corresponding system variables that have `_commit_` in their name.

For example, if you have reconfigured HashiCorp Vault to listen on port 8201 rather than the default 8200, reconfigure `keyring_hashicorp` like this:

```
mysql> SET GLOBAL keyring_hashicorp_server_url = 'https://127.0.0.1:8201';
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT keyring_hashicorp_update_config();
+--------------------------------------+
| keyring_hashicorp_update_config()    |
+--------------------------------------+
| Configuration update was successful. |
+--------------------------------------+
1 row in set (0.03 sec)
```

If the plugin is not able to connect to HashiCorp Vault during initialization or reconfiguration and there was no existing connection, the `_commit_` system variables are set to 'Not committed' for string-valued variables, and OFF for Boolean-valued variables. If the plugin is not able to connect but there was an existing connection, that connection remains active and the `_commit_` variables reflect the values used for it.
Migrating Keys Between Keyring Keystores

Note
If you do not set the mandatory system variables at server startup, or if some other plugin initialization error occurs, initialization fails. In this case, you can use the runtime reconfiguration procedure to initialize the plugin without restarting the server.

For additional information about the keyring_hashicorp plugin-specific system variables and UDF, see Section 6.4.12, “Keyring System Variables”, and Section 6.4.10, “Plugin-Specific Keyring Key-Management Functions”.

6.4.7 Migrating Keys Between Keyring Keystores

The MySQL server supports an operational mode that enables migration of keys between underlying keyring keystores. This enables DBAs to switch a MySQL installation from one keyring plugin to another. A migration server (that is, a server started in key migration mode) does not accept client connections. Instead, it runs only long enough to migrate keys, then exits. A migration server reports errors to the console (the standard error output).

It is possible to perform offline or online key migration:

- If you are sure that no running server on the local host is using the source or destination keystore, an offline migration is possible. In this case, the migration server can modify the keystores without possibility of a running server modifying keystore content during the migration.

- If a running server on the local host is using the source or destination keystore, an online migration must be performed. In this case, the migration server connects to the running server and instructs it to pause keyring operations while key migration is in progress.

The result of a key migration operation is that the destination keystore contains the keys it had prior to the migration, plus the keys from the source keystore. The source keystore is the same before and after the migration because keys are copied, not moved. If a key to be copied already exists in the destination keystore, an error occurs and the destination keystore is restored to its premigration state.

The user who invokes the server in key-migration mode must not be the root operating system user, and must have permission to read and write the keyring files.

To perform a key migration operation, determine which key migration options are needed. Migration options indicate which keyring plugins are involved, and whether to perform an offline or online migration:

- To indicate the source and destination keyring plugins, specify these options:
  
  - --keyring-migration-source: The source keyring plugin that manages the keys to be migrated.

  - --keyring-migration-destination: The destination keyring plugin to which the migrated keys are to be copied.

  These options tell the server to run in key migration mode. Both options are mandatory for all key migration operations. The source and destination plugins must differ, and the migration server must support both plugins.

- For an offline migration, no additional key migration options are needed.

  Warning
  Do not perform offline migration involving a keystore that is in use by a running server.

- For an online migration, some running server currently is using the source or destination keystore. Specify the key migration options that indicate how to connect to the running server. This is
Migrating Keys Between Keyring Keystores

necessary so that the migration server can connect to the running server and tell it to pause keyring use during the migration operation.

Use of any of the following options signifies an online migration:

- **--keyring-migration-host**: The host where the running server is located. This is always the local host.
- **--keyring-migration-user, --keyring-migration-password**: The user name and password for the account to use to connect to the running server.
- **--keyring-migration-port**: For TCP/IP connections, the port number to connect to on the running server.
- **--keyring-migration-socket**: For Unix socket file or Windows named pipe connections, the socket file or named pipe to connect to on the running server.

For additional details about the key migration options, see Section 6.4.11, “Keyring Command Options”.

Start the migration server with the key migration options determined as just described, possibly with other options. Keep the following considerations in mind:

- Other server options might be required, such as other configuration parameters for the two keyring plugins. For example, if `keyring_file` is one of the plugins, you must set the `keyring_file_data` system variable if the keyring data file location is not the default location. Other non-keyring options may be required as well. One way to specify these options is by using `--defaults-file` to name an option file that contains the required options.

- If you invoke the migration server from a system account different from that normally used to run MySQL, it might create keyring directories or files that are inaccessible to the server during normal operation. Suppose that `mysqld` normally runs as the `mysql` operating system user, but you invoke the migration server while logged in as `isabel`. Any new directories or files created by the migration server will be owned by `isabel`. Subsequent startup will fail when a server run as the `mysql` operating system user attempts to access file system objects owned by `isabel`.

To avoid this issue, start the migration server as the `root` operating system user and provide a `--user=user_name` option, where `user_name` is the system account normally used to run MySQL.

- The migration server expects path name option values to be full paths. Relative path names may not be resolved as you expect.

Example command line for offline key migration:

```
mysqld --defaults-file=/usr/local/mysql/etc/my.cnf
--keyring-migration-source=keyring_file.so
--keyring-migration-destination=keyring_encrypted_file.so
--keyring_encrypted_file_password=password
```

Example command line for online key migration:

```
mysqld --defaults-file=/usr/local/mysql/etc/my.cnf
--keyring-migration-source=keyring_file.so
--keyring-migration-destination=keyring_encrypted_file.so
--keyring_encrypted_file_password=password
--keyring-migration-host=localhost
--keyring-migration-user=root
--keyring-migration-password=root_password
```

The key migration server performs the migration operation as follows:

1. (Online migration only) Connect to the running server using the connection options. The account used to connect must have the privileges required to modify the global keyring_operations
system variable (`ENCRYPTION_KEY_ADMIN` in addition to either `SYSTEM_VARIABLES_ADMIN` or `SUPER`).

2. (Online migration only) Disable `keyring_operations` on the running server. (The running server must support `keyring_operations`.)

3. Load the source and destination keyring plugins.

4. Copy keys from the source keyring to the destination keyring.

5. Unload the keyring plugins.

6. (Online migration only) Enable `keyring_operations` on the running server.

7. (Online migration only) Disconnect from the running server.

8. Exit.

If an error occurs during key migration, any keys that were copied to the destination plugin are removed, leaving the destination keystore unchanged.

**Important**

For an online migration operation, the migration server takes care of enabling and disabling `keyring_operations` on the running server. However, if the migration server exits abnormally (for example, if someone forcibly terminates it), it is possible that `keyring_operations` will not have been re-enabled on the running server, leaving it unable to perform keyring operations. In this case, it may be necessary to connect to the running server and enable `keyring_operations` manually.

After a successful online key migration operation, the running server might need to be restarted:

- If the running server was using the source keystore, it need not be restarted after the migration.
- If the running server was using the source keystore before the migration but should use the destination keystore after the migration, it must be reconfigured to use the destination keyring plugin and restarted.
- If the running server was using the destination keystore and will continue to use it, it should be restarted after the migration to load all keys migrated into the destination keystore.

**Note**

MySQL server key migration mode supports pausing a single running server. To perform a key migration if multiple key servers are using the keystores involved, use this procedure:

1. Connect to each running server manually and set `keyring_operations=OFF`.
2. Use the migration server to perform an offline key migration.
3. Connect to each running server manually and set `keyring_operations=ON`.

All running servers must support the `keyring_operations=ON` system variable.

### 6.4.8 Supported Keyring Key Types and Lengths

MySQL Keyring supports keys of different types (encryption algorithms) and lengths:
• The available key types depend on which keyring plugin is installed.

• The permitted key lengths are subject to multiple factors:
  
  • General keyring UDF interface limits (for keys managed using one of the keyring UDFs described in Section 6.4.9, “General-Purpose Keyring Key-Management Functions”), or limits from back end implementations. These length limits can vary by key operation type.

  • In addition to the general limits, individual plugins may impose restrictions on key lengths per key type.

Table 6.15, “General Keyring Key Length Limits” shows the general key length limits. (The lower limits for keyring_aws are imposed by the AWS KMS interface, not the keyring UDFs.) Table 6.16, “Keyring Plugin Key Types and Lengths” shows for each keyring plugin the key types it permits, as well as any plugin-specific key-length restrictions.

Table 6.15 General Keyring Key Length Limits

<table>
<thead>
<tr>
<th>Key Operation</th>
<th>Maximum Key Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate key</td>
<td>16,384 bytes (2,048 prior to MySQL 8.0.18); 1,024 for keyring_aws</td>
</tr>
<tr>
<td>Store key</td>
<td>16,384 bytes (2,048 prior to MySQL 8.0.18); 4,096 for keyring_aws</td>
</tr>
<tr>
<td>Fetch key</td>
<td>16,384 bytes (2,048 prior to MySQL 8.0.18); 4,096 for keyring_aws</td>
</tr>
</tbody>
</table>

Table 6.16 Keyring Plugin Key Types and Lengths

<table>
<thead>
<tr>
<th>Plugin Name</th>
<th>Permitted Key Type</th>
<th>Plugin-Specific Length Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyring_encrypted_file</td>
<td>AES</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>DSA</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>RSA</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>SECRET</td>
<td>None</td>
</tr>
<tr>
<td>keyring_file</td>
<td>AES</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>DSA</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>RSA</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>SECRET</td>
<td>None</td>
</tr>
<tr>
<td>keyring_okv</td>
<td>AES</td>
<td>16, 24, or 32 bytes</td>
</tr>
<tr>
<td></td>
<td>SECRET</td>
<td>None</td>
</tr>
<tr>
<td>keyring_aws</td>
<td>AES</td>
<td>16, 24, or 32 bytes</td>
</tr>
<tr>
<td></td>
<td>SECRET</td>
<td>None</td>
</tr>
<tr>
<td>keyring_hashicorp</td>
<td>AES</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>DSA</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>RSA</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>SECRET</td>
<td>None</td>
</tr>
</tbody>
</table>

The SECRET key type, available as of MySQL 8.0.19, is intended for general-purpose storage of sensitive data using the MySQL keyring, and is supported by all keyring plugins. The keyring encrypts and decrypts SECRET data as a byte stream upon storage and retrieval.

Example keyring operations involving the SECRET key type:
6.4.9 General-Purpose Keyring Key-Management Functions

MySQL Server supports a keyring service that enables internal server components and plugins to securely store sensitive information for later retrieval.

MySQL Server also includes an SQL interface for keyring key management, implemented as a set of general-purpose user-defined functions (UDFs) that access the functions provided by the internal keyring service. The keyring UDFs are contained in a plugin library file, which also contains a keyring_udf plugin that must be enabled prior to UDF invocation. For these UDFs to be used, a keyring plugin such as keyring_file or keyring_okv must be enabled.

The UDFs described here are general purpose and intended for use with any keyring plugin. A given keyring plugin might have UDFs of its own that are intended for use only with that plugin; see Section 6.4.10, “Plugin-Specific Keyring Key-Management Functions”.

The following sections provide installation instructions for the keyring UDFs and demonstrate how to use them. For information about the keyring service functions invoked by the UDFs, see The Keyring Service. For general keyring information, see Section 6.4, “The MySQL Keyring”.

- Installing or Uninstalling General-Purpose Keyring Functions
- Using General-Purpose Keyring Functions
- General-Purpose Keyring Function Reference

Installing or Uninstalling General-Purpose Keyring Functions

This section describes how to install or uninstall the keyring user-defined functions (UDFs), which are implemented in a plugin library file that also contains a keyring_udf plugin. For general information about installing or uninstalling plugins and UDFs, see Installing and Uninstalling Plugins, and Installing and Uninstalling User-Defined Functions.

The keyring UDFs enable keyring key management operations, but the keyring_udf plugin must also be installed because the UDFs will not work correctly without it. Attempts to use the UDFs without the keyring_udf plugin result in an error.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the plugin_dir system variable). If necessary, configure the plugin directory location by setting the value of plugin_dir at server startup.

The plugin library file base name is keyring_udf. The file name suffix differs per platform (for example, .so for Unix and Unix-like systems, .dll for Windows).

To install the keyring_udf plugin and the UDFs, use the INSTALL PLUGIN and CREATE FUNCTION statements (adjust the .so suffix for your platform as necessary):

```
INSTALL PLUGIN keyring_udf SONAME 'keyring_udf.so';
CREATE FUNCTION keyring_key_generate RETURNS INTEGER
  SONAME 'keyring_udf.so';
CREATE FUNCTION keyring_key_fetch RETURNS STRING
  SONAME 'keyring_udf.so';
CREATE FUNCTION keyring_key_length_fetch RETURNS INTEGER
  SONAME 'keyring_udf.so';
CREATE FUNCTION keyring_key_type_fetch RETURNS STRING
  SONAME 'keyring_udf.so';
CREATE FUNCTION keyring_key_store RETURNS INTEGER
  SONAME 'keyring_udf.so';
```
SONAME 'keyring_udf.so';
CREATE FUNCTION keyring_key_remove RETURNS INTEGER
SONAME 'keyring_udf.so';

If the plugin and UDFs are used on a master replication server, install them on all slave servers as well to avoid replication issues.

Once installed as just described, the plugin and UDFs remain installed until uninstalled. To remove them, use the UNINSTALL PLUGIN and DROP FUNCTION statements:

UNINSTALL PLUGIN keyring_udf;
DROP FUNCTION keyring_key_generate;
DROP FUNCTION keyring_key_fetch;
DROP FUNCTION keyring_key_length_fetch;
DROP FUNCTION keyring_key_type_fetch;
DROP FUNCTION keyring_key_store;
DROP FUNCTION keyring_key_remove;

Using General-Purpose Keyring Functions

Before using the keyring user-defined functions (UDFs), install them according to the instructions provided in Installing or Uninstalling General-Purpose Keyring Functions.

The keyring UDFs are subject to these constraints:

• To use any keyring UDF, the keyring_udf plugin must be enabled. Otherwise, an error occurs:

ERROR 1123 (HY000): Can't initialize function 'keyring_key_generate';
This function requires keyring_udf plugin which is not installed.
Please install

To install the keyring_udf plugin, see Installing or Uninstalling General-Purpose Keyring Functions.

• The keyring UDFs invoke keyring service functions (see The Keyring Service). The service functions in turn use whatever keyring plugin is installed (for example, keyring_file or keyring_okv). Therefore, to use any keyring UDF, some underlying keyring plugin must be enabled. Otherwise, an error occurs:

ERROR 3188 (HY000): Function 'keyring_key_generate' failed because underlying keyring service returned an error. Please check if a keyring plugin is installed and that provided arguments are valid for the keyring you are using.

To install a keyring plugin, see Section 6.4.1, “Keyring Plugin Installation”.

• To use any keyring UDF, a user must possess the global EXECUTE privilege. Otherwise, an error occurs:

ERROR 1123 (HY000): Can't initialize function 'keyring_key_generate';
The user is not privileged to execute this function. User needs to have EXECUTE

To grant the global EXECUTE privilege to a user, use this statement:

GRANT EXECUTE ON *.* TO user;

Alternatively, should you prefer to avoid granting the global EXECUTE privilege while still permitting users to access specific key-management operations, “wrapper” stored programs can be defined (a technique described later in this section).

• A key stored in the keyring by a given user can be manipulated later only by the same user. That is, the value of the CURRENT_USER() function at the time of key manipulation must have the same value as when the key was stored in the keyring. (This constraint rules out the use of the keyring UDFs for manipulation of instance-wide keys, such as those created by InnoDB to support tablespace encryption.)
To enable multiple users to perform operations on the same key, “wrapper” stored programs can be defined (a technique described later in this section).

- Keyring UDFs support the key types and lengths supported by the underlying keyring plugin. For information about keys specific to a particular keyring plugin, see Section 6.4.8, “Supported Keyring Key Types and Lengths”.

To create a new random key and store it in the keyring, call `keyring_key_generate()`, passing to it an ID for the key, along with the key type (encryption method) and its length in bytes. The following call creates a 2,048-bit DSA-encrypted key named `MyKey`:

```sql
mysql> SELECT keyring_key_generate('MyKey', 'DSA', 256);
```

A return value of 1 indicates success. If the key cannot be created, the return value is `NULL` and an error occurs. One reason this might be is that the underlying keyring plugin does not support the specified combination of key type and key length; see Section 6.4.8, “Supported Keyring Key Types and Lengths”.

To be able to check the return type regardless of whether an error occurs, use `SELECT ... INTO @var_name` and test the variable value:

```sql
mysql> SELECT keyring_key_generate('', '', -1) INTO @x;
ERROR 3188 (HY000): Function 'keyring_key_generate' failed because underlying keyring service returned an error. Please check if a keyring plugin is installed and that provided arguments are valid for the keyring you are using.
```

```sql
mysql> SELECT @x;
+------+
| @x   |
+------+
| NULL |
```

This technique also applies to other keyring UDFs that for failure return a value and an error.

The ID passed to `keyring_key_generate()` provides a means by which to refer to the key in subsequent UDF calls. For example, use the key ID to retrieve its type as a string or its length in bytes as an integer:

```sql
mysql> SELECT keyring_key_type_fetch('MyKey');
| keyring_key_type_fetch('MyKey') |
| DSA                             |
```

```sql
mysql> SELECT keyring_key_length_fetch('MyKey');
| keyring_key_length_fetch('MyKey') |
| 256                             |
```

To retrieve a key value, pass the key ID to `keyring_key_fetch()`. The following example uses `HEX()` to display the key value because it may contain nonprintable characters. The example also uses a short key for brevity, but be aware that longer keys provide better security:
General-Purpose Keyring Key-Management Functions

Keyring UDFs treat key IDs, types, and values as binary strings, so comparisons are case-sensitive. For example, IDs of `MyKey` and `mykey` refer to different keys.

To remove a key, pass the key ID to `keyring_key_remove()`:

```sql
mysql> SELECT keyring_key_remove('MyKey');
```

To obfuscate and store a key that you provide, pass the key ID, type, and value to `keyring_key_store()`:

```sql
mysql> SELECT keyring_key_store('AES_key', 'AES', 'Secret string');
```

As indicated previously, a user must have the global `EXECUTE` privilege to call keyring UDFs, and the user who stores a key in the keyring initially must be the same user who performs subsequent operations on the key later, as determined from the `CURRENT_USER()` value in effect for each UDF call. To permit key operations to users who do not have the global `EXECUTE` privilege or who may not be the key “owner,” use this technique:

1. Define “wrapper” stored programs that encapsulate the required key operations and have a `DEFINER` value equal to the key owner.

2. Grant the `EXECUTE` privilege for specific stored programs to the individual users who should be able to invoke them.

3. If the operations implemented by the wrapper stored programs do not include key creation, create any necessary keys in advance, using the account named as the `DEFINER` in the stored program definitions.

This technique enables keys to be shared among users and provides to DBAs more fine-grained control over who can do what with keys, without having to grant global privileges.

The following example shows how to set up a shared key named `SharedKey` that is owned by the DBA, and a `get_shared_key()` stored function that provides access to the current key value. The value can be retrieved by any user with the `EXECUTE` privilege for that function, which is created in the `key_schema` schema.

From a MySQL administrative account (`'root'@'localhost'` in this example), create the administrative schema and the stored function to access the key:

```sql
CREATE SCHEMA key_schema;
CREATE DEFINER = 'root'@'localhost'
FUNCTION key_schema.get_shared_key()
RETURNS BLOB READS SQL DATA
RETURN keyring_key_fetch('SharedKey');
```
From the administrative account, ensure that the shared key exists:

```sql
mysql> SELECT keyring_key_generate('SharedKey', 'DSA', 8);
+---------------------------------------------+
| keyring_key_generate('SharedKey', 'DSA', 8) |
+---------------------------------------------+
|                                           1 |
+---------------------------------------------+
```

From the administrative account, create an ordinary user account to which key access is to be granted:

```sql
mysql> CREATE USER 'key_user'@'localhost'
  IDENTIFIED BY 'key_user_pwd';
```

From the `key_user` account, verify that, without the proper `EXECUTE` privilege, the new account cannot access the shared key:

```sql
mysql> SELECT HEX(key_schema.get_shared_key());
ERROR 1370 (42000): execute command denied to user 'key_user'@'localhost'
  for routine 'key_schema.get_shared_key'
```

From the administrative account, grant `EXECUTE` to `key_user` for the stored function:

```sql
mysql> GRANT EXECUTE ON FUNCTION key_schema.get_shared_key
  TO 'key_user'@'localhost';
```

From the `key_user` account, verify that the key is now accessible:

```sql
mysql> SELECT HEX(key_schema.get_shared_key());
+----------------------------------+
| HEX(key_schema.get_shared_key()) |
+----------------------------------+
| 9BAFB9E75C6EB013                 |
+----------------------------------+
```

---

**General-Purpose Keyring Function Reference**

For each general-purpose keyring user-defined function (UDF), this section describes its purpose, calling sequence, and return value. For information about the conditions under which these UDFs can be invoked, see Using General-Purpose Keyring Functions.

- **keyring_key_fetch(key_id)**

  Given a key ID, deobfuscates and returns the key value.

  Arguments:

  - **key_id**: A string that specifies the key ID.

  Return value:

  Returns the key value as a string for success, `NULL` if the key does not exist, or `NULL` and an error for failure.

  **Note**

  Key values retrieved using `keyring_key_fetch()` are subject to the general keyring UDF limits described in Section 6.4.8, “Supported Keyring Key Types and Lengths”. A key value longer than that length can be stored using a keyring service function (see The Keyring Service), but if retrieved using `keyring_key_fetch()` is truncated to the general keyring UDF limit.

  **Example:**

  ```sql
  mysql> SELECT keyring_key_generate('RSA_key', 'RSA', 16);
  +--------------------------------------------+
  |                                           |
  +--------------------------------------------+
  ```
The example uses `HEX()` to display the key value because it may contain nonprintable characters. The example also uses a short key for brevity, but be aware that longer keys provide better security.

- `keyring_key_generate(key_id, key_type, key_length)`

  Generates a new random key with a given ID, type, and length, and stores it in the keyring. The type and length values must be consistent with the values supported by the underlying keyring plugin. See Section 6.4.8, “Supported Keyring Key Types and Lengths”.

  Arguments:

  - `key_id`: A string that specifies the key ID.
  - `key_type`: A string that specifies the key type.
  - `key_length`: An integer that specifies the key length in bytes.

  Return value:

  Returns 1 for success, or `NULL` and an error for failure.

  Example:

  ```
  mysql> SELECT keyring_key_generate('RSA_key', 'RSA', 384);
  +-------------------------------------------------+
  | keyring_key_generate('RSA_key', 'RSA', 384) |
  +-------------------------------------------------+
  |                                           1 |
  +-------------------------------------------------+
  ```

- `keyring_key_length_fetch(key_id)`

  Given a key ID, returns the key length.

  Arguments:

  - `key_id`: A string that specifies the key ID.

  Return value:

  Returns the key length in bytes as an integer for success, `NULL` if the key does not exist, or `NULL` and an error for failure.

  Example:
See the description of `keyring_key_fetch()`.

- **keyring_key_remove(key_id)**

Removes the key with a given ID from the keyring.

Arguments:
- **key_id**: A string that specifies the key ID.

Return value:

Returns 1 for success, or `NULL` for failure.

Example:

```sql
mysql> SELECT keyring_key_remove('AES_key');
+-------------------------------+
| keyring_key_remove('AES_key') |
| 1                             |
+-------------------------------+
```

- **keyring_key_store(key_id, key_type, key)**

Obfuscates and stores a key in the keyring.

Arguments:
- **key_id**: A string that specifies the key ID.
- **key_type**: A string that specifies the key type.
- **key**: A string that specifies the key value.

Return value:

Returns 1 for success, or `NULL` and an error for failure.

Example:

```sql
mysql> SELECT keyring_key_store('new key', 'DSA', 'My key value');
+-----------------------------------------------------+
| keyring_key_store('new key', 'DSA', 'My key value') |
| 1                                                   |
+-----------------------------------------------------+
```

- **keyring_key_type_fetch(key_id)**

Given a key ID, returns the key type.

Arguments:
- **key_id**: A string that specifies the key ID.

Return value:

Returns the key type as a string for success, `NULL` if the key does not exist, or `NULL` and an error for failure.

Example:

See the description of `keyring_key_fetch()`.
6.4.10 Plugin-Specific Keyring Key-Management Functions

For each keyring plugin-specific user-defined function (UDF), this section describes its purpose, calling sequence, and return value. For information about general-purpose keyring UDFs, see Section 6.4.9, “General-Purpose Keyring Key-Management Functions”.

- **keyring_aws_rotate_cmk()**
  
  Associated keyring plugin: **keyring_aws**

  **keyring_aws_rotate_cmk()** rotates the customer master key (CMK). Rotation changes only the key that AWS KMS uses for subsequent data key-encryption operations. AWS KMS maintains previous CMK versions, so keys generated using previous CMKs remain decryptable after rotation.

  Rotation changes the CMK value used inside AWS KMS but does not change the ID used to refer to it, so there is no need to change the **keyring_aws_cmk_id** system variable after calling **keyring_aws_rotate_cmk()**.

  This UDF requires the **SUPER** privilege.

  Arguments:
  None.

  Return value:
  Returns 1 for success, or **NULL** and an error for failure.

- **keyring_aws_rotate_keys()**
  
  Associated keyring plugin: **keyring_aws**

  **keyring_aws_rotate_keys()** rotates keys stored in the **keyring_aws** storage file named by the **keyring_aws_data_file** system variable. Rotation sends each key stored in the file to AWS KMS for re-encryption using the value of the **keyring_aws_cmk_id** system variable as the CMK value, and stores the new encrypted keys in the file.

  **keyring_aws_rotate_keys()** is useful for key re-encryption under these circumstances:

  - After rotating the CMK; that is, after invoking the **keyring_aws_rotate_cmk()** UDF
  - After changing the **keyring_aws_cmk_id** system variable to a different key value

  This UDF requires the **SUPER** privilege.

  Arguments:
  None.

  Return value:
  Returns 1 for success, or **NULL** and an error for failure.

- **keyring_hashicorp_update_config()**
  
  Associated keyring plugin: **keyring_hashicorp**

  When invoked, the **keyring_hashicorp_update_config()** UDF causes **keyring_hashicorp** to perform a runtime reconfiguration, as described in **keyring_hashicorp Configuration**.

  This UDF requires the **SYSTEM_VARIABLES_ADMIN** privilege because it modifies global system variables.
Arguments:

None.

Return value:

Returns the string 'Configuration update was successful.' for success, or 'Configuration update failed.' for failure.

6.4.11 Keyring Command Options

MySQL supports the following keyring-related command-line options:

• \texttt{--keyring-migration-destination=plugin}

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td>\texttt{--keyring-migration-destination=plugin_name}</td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.4</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
</tbody>
</table>

The destination keyring plugin for key migration. See Section 6.4.7, “Migrating Keys Between Keyring Keystores”. The format and interpretation of the option value is the same as described for the \texttt{--keyring-migration-source} option.

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{--keyring-migration-source} and \texttt{--keyring-migration-destination} are mandatory for all keyring migration operations. The source and destination plugins must differ, and the migration server must support both plugins.</td>
</tr>
</tbody>
</table>

• \texttt{--keyring-migration-host=host\_name}

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td>\texttt{--keyring-migration-host=host_name}</td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.4</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td>localhost</td>
</tr>
</tbody>
</table>

The host location of the running server that is currently using one of the key migration keystores. See Section 6.4.7, “Migrating Keys Between Keyring Keystores”. Migration always occurs on the local host, so the option always specifies a value for connecting to a local server, such as localhost, 127.0.0.1, ::1, or the local host IP address or host name.

• \texttt{--keyring-migration-password[=password]}

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td>\texttt{--keyring-migration-password[=password]}</td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.4</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
</tbody>
</table>

The password for connecting to the running server that is currently using one of the key migration keystores. See Section 6.4.7, “Migrating Keys Between Keyring Keystores”. If you omit the \texttt{password} value following the option name on the command line, the server prompts for one.
Specifying a password on the command line should be considered insecure. See Section 2.2.1, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line. In this case, the file should have a restrictive mode and be accessible only to the account used to run the migration server.

- **--keyring-migration-port=port_num**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--keyring-migration-port=port_num</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.4</td>
</tr>
<tr>
<td>Type</td>
<td>Numeric</td>
</tr>
<tr>
<td>Default Value</td>
<td>3306</td>
</tr>
</tbody>
</table>

For TCP/IP connections, the port number for connecting to the running server that is currently using one of the key migration keystores. See Section 6.4.7, “Migrating Keys Between Keyring Keystores”.

- **--keyring-migration-socket=path**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td>`--keyring-migration-socket={file_name</td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.4</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
</tbody>
</table>

For Unix socket file or Windows named pipe connections, the socket file or named pipe for connecting to the running server that is currently using one of the key migration keystores. See Section 6.4.7, “Migrating Keys Between Keyring Keystores”.

- **--keyring-migration-source=plugin**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--keyring-migration-source=plugin_name</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.4</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
</tbody>
</table>

The source keyring plugin for key migration. See Section 6.4.7, “Migrating Keys Between Keyring Keystores”.

The option value is similar to that for **--plugin-load**, except that only one plugin library can be specified. The value is given as name(plugin_library) or plugin_library. The **name** is the name of a plugin to load, and **plugin_library** is the name of the library file that contains the plugin code. If the plugin library is named without any preceding plugin name, the server loads all plugins in the library. The server looks for plugin library files in the directory named by the **plugin_dir** system variable.

**Note**

**--keyring-migration-source** and **--keyring-migration-destination** are mandatory for all keyring migration operations. The source and destination plugins must differ, and the migration server must support both plugins.
### Keyring System Variables

MySQL Keyring plugins support the following system variables. Use them to configure keyring plugin operation. These variables are unavailable unless the appropriate keyring plugin is installed (see Section 6.4.1, “Keyring Plugin Installation”).

- **--keyring-migration-user=user_name**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td>--keyring-migration-user=user_name</td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.4</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
</tbody>
</table>

The user name for connecting to the running server that is currently using one of the key migration keystores. See Section 6.4.7, “Migrating Keys Between Keyring Keystores”.

### 6.4.12 Keyring System Variables

**keyring_aws_cmk_id**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td>--keyring-aws-cmk-id=value</td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td>keyring_aws_cmk_id</td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
</tbody>
</table>

The customer master key (CMK) ID obtained from the AWS KMS server and used by the keyring_aws plugin. This variable is unavailable unless that plugin is installed, but if it is installed, a value for this variable is mandatory.

- **keyring_aws_conf_file**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td>--keyring-aws-conf-file=file_name</td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td>keyring_aws_conf_file</td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>No</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>File name</td>
</tr>
<tr>
<td>Default Value</td>
<td>platform specific</td>
</tr>
</tbody>
</table>

The location of the configuration file for the keyring_aws keyring plugin. This variable is unavailable unless that plugin is installed.

At plugin startup, keyring_aws reads the AWS secret access key ID and key from the configuration file. For the keyring_aws plugin to start successfully, the configuration file must exist and contain valid secret access key information, initialized as described in Section 6.4.5, “Using the keyring_aws Amazon Web Services Keyring Plugin”. 
The default file name is `keyring_aws_conf`, located in the default keyring file directory. The location of this default directory is the same as for the `keyring_file_data` system variable. See the description of that variable for details, as well as for considerations to take into account if you create the directory manually.

- **keyring_aws_data_file**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--keyring-aws-data-file</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>keyring_aws_data_file</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>No</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>File name</td>
</tr>
<tr>
<td>Default Value</td>
<td><code>platform specific</code></td>
</tr>
</tbody>
</table>

The location of the storage file for the `keyring_aws` keyring plugin. This variable is unavailable unless that plugin is installed.

At plugin startup, if the value assigned to `keyring_aws_data_file` specifies a file that does not exist, the `keyring_aws` plugin attempts to create it (as well as its parent directory, if necessary). If the file does exist, `keyring_aws` reads any encrypted keys contained in the file into its in-memory cache. `keyring_aws` does not cache unencrypted keys in memory.

The default file name is `keyring_aws_data`, located in the default keyring file directory. The location of this default directory is the same as for the `keyring_file_data` system variable. See the description of that variable for details, as well as for considerations to take into account if you create the directory manually.

- **keyring_aws_region**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--keyring-aws-region=value</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>keyring_aws_region</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Default Value</td>
<td><code>us-east-1</code></td>
</tr>
<tr>
<td>Valid Values</td>
<td><code>ap-northeast-1</code></td>
</tr>
<tr>
<td></td>
<td><code>ap-northeast-2</code></td>
</tr>
<tr>
<td></td>
<td><code>ap-south-1</code></td>
</tr>
<tr>
<td></td>
<td><code>ap-southeast-1</code></td>
</tr>
<tr>
<td></td>
<td><code>ap-southeast-2</code></td>
</tr>
<tr>
<td></td>
<td><code>eu-central-1</code></td>
</tr>
</tbody>
</table>
Keyring System Variables

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>eu-west-1</td>
<td></td>
</tr>
<tr>
<td>sa-east-1</td>
<td></td>
</tr>
<tr>
<td>us-east-1</td>
<td></td>
</tr>
<tr>
<td>us-west-1</td>
<td></td>
</tr>
<tr>
<td>us-west-2</td>
<td></td>
</tr>
</tbody>
</table>

The AWS region.

- **keyring_encrypted_file_data**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--keyring-encrypted-file-data=file_name</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>keyring_encrypted_file_data</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><code>SET_VAR</code> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>File name</td>
</tr>
<tr>
<td>Default Value</td>
<td><code>platform specific</code></td>
</tr>
</tbody>
</table>

The path name of the data file used for secure data storage by the `keyring_encrypted_file` plugin. This variable is unavailable unless that plugin is installed. The file location should be in a directory considered for use only by keyring plugins. For example, do not locate the file under the data directory.

Keyring operations are transactional: The `keyring_encrypted_file` plugin uses a backup file during write operations to ensure that it can roll back to the original file if an operation fails. The backup file has the same name as the value of the `keyring_encrypted_file_data` system variable with a suffix of `.backup`.

Do not use the same `keyring_encrypted_file` data file for multiple MySQL instances. Each instance should have its own unique data file.

The default file name is `keyring_encrypted`, located in a directory that is platform specific and depends on the value of the `INSTALL_LAYOUT` CMake option, as shown in the following table. To specify the default directory for the file explicitly if you are building from source, use the `INSTALL_MYSQLKEYRINGDIR` CMake option.

<table>
<thead>
<tr>
<th><code>INSTALL_LAYOUT</code> Value</th>
<th>Default <code>keyring_encrypted_file_data</code> Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEB, RPM, SVR4</td>
<td><code>/var/lib/mysql-keyring/keyring_encrypted</code></td>
</tr>
<tr>
<td>Otherwise</td>
<td><code>keyring/keyring_encrypted</code> under the <code>CMAKE_INSTALL_PREFIX</code> value</td>
</tr>
</tbody>
</table>

At plugin startup, if the value assigned to `keyring_encrypted_file_data` specifies a file that does not exist, the `keyring_encrypted_file` plugin attempts to create it (as well as its parent directory, if necessary).

If you create the directory manually, it should have a restrictive mode and be accessible only to the account used to run the MySQL server. For example, on Unix and Unix-like systems, to use the...
Keyring System Variables

`/usr/local/mysql/mysql-keyring` directory, the following commands (executed as root) create the directory and set its mode and ownership:

```
cd /usr/local/mysql
mkdir mysql-keyring
chmod 750 mysql-keyring
chown mysql mysql-keyring
chgrp mysql mysql-keyring
```

If the `keyring_encrypted_file` plugin cannot create or access its data file, it writes an error message to the error log. If an attempted runtime assignment to `keyring_encrypted_file_data` results in an error, the variable value remains unchanged.

---

**Important**

Once the `keyring_encrypted_file` plugin has created its data file and started to use it, it is important not to remove the file. Loss of the file will cause data encrypted using its keys to become inaccessible. (It is permissible to rename or move the file, as long as you change the value of `keyring_encrypted_file_data` to match.)

---

• `keyring_encrypted_file_password`

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--keyring-encrypted-file-password=password</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>keyring_encrypted_file_password</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>SET_VAR</strong> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
</tbody>
</table>

The password used by the `keyring_encrypted_file` plugin. This variable is unavailable unless that plugin is installed. The password is mandatory for plugin operation; if not specified at server startup, `keyring_encrypted_file` initialization fails.

If this variable is specified in an option file, the file should have a restrictive mode and be accessible only to the account used to run the MySQL server.

---

**Important**

Once the `keyring_encrypted_file_password` value has been set, changing it does not rotate the keyring password and could make the server inaccessible. If an incorrect password is provided, the `keyring_encrypted_file` plugin cannot load keys from the encrypted keyring file.

The password value cannot be displayed at runtime with `SHOW VARIABLES` or the Performance Schema `global_variables` table because the display value is obfuscated.

---

• `keyring_file_data`

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--keyring-file-data=file_name</code></td>
</tr>
<tr>
<td>System Variable</td>
<td><code>keyring_file_data</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
</tbody>
</table>
Keyring System Variables

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>File name</td>
</tr>
<tr>
<td>Default Value</td>
<td><strong>platform specific</strong></td>
</tr>
</tbody>
</table>

The path name of the data file used for secure data storage by the `keyring_file` plugin. This variable is unavailable unless that plugin is installed. The file location should be in a directory considered for use only by keyring plugins. For example, do not locate the file under the data directory.

Keyring operations are transactional: The `keyring_file` plugin uses a backup file during write operations to ensure that it can roll back to the original file if an operation fails. The backup file has the same name as the value of the `keyring_file_data` system variable with a suffix of `.backup`.

Do not use the same `keyring_file` data file for multiple MySQL instances. Each instance should have its own unique data file.

The default file name is `keyring`, located in a directory that is platform specific and depends on the value of the `INSTALL_LAYOUT` CMake option, as shown in the following table. To specify the default directory for the file explicitly if you are building from source, use the `INSTALL_MYSQLKEYRINGDIR` CMake option.

<table>
<thead>
<tr>
<th><code>INSTALL_LAYOUT</code> Value</th>
<th>Default <code>keyring_file_data</code> Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEB, RPM, SVR4</td>
<td><code>/var/lib/mysql-keyring/keyring</code></td>
</tr>
<tr>
<td>Otherwise</td>
<td><code>keyring/keyring</code> under the <code>CMAKE_INSTALL_PREFIX</code> value</td>
</tr>
</tbody>
</table>

At plugin startup, if the value assigned to `keyring_file_data` specifies a file that does not exist, the `keyring_file` plugin attempts to create it (as well as its parent directory, if necessary).

If you create the directory manually, it should have a restrictive mode and be accessible only to the account used to run the MySQL server. For example, on Unix and Unix-like systems, to use the `/usr/local/mysql/mysql-keyring` directory, the following commands (executed as root) create the directory and set its mode and ownership:

```bash
cd /usr/local/mysql
mkdir mysql-keyring
chmod 750 mysql-keyring
chown mysql mysql-keyring
chgrp mysql mysql-keyring
```

If the `keyring_file` plugin cannot create or access its data file, it writes an error message to the error log. If an attempted runtime assignment to `keyring_file_data` results in an error, the variable value remains unchanged.

**Important**

Once the `keyring_file` plugin has created its data file and started to use it, it is important not to remove the file. For example, InnoDB uses the file to store the master key used to decrypt the data in tables that use InnoDB tablespace encryption; see InnoDB Data-at-Rest Encryption. Loss of the file will cause data in such tables to become inaccessible. (It is permissible to rename or move the file, as long as you change the value of `keyring_file_data` to match.) It is recommended that you create a separate backup of the keyring data file immediately after you create the first encrypted table and before and after master key rotation.
### Keyring System Variables

- **keyring_hashicorp_auth_path**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--keyring-hashicorp-auth-path=value</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.18</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>keyring_hashicorp_auth_path</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td><code>/v1/auth/approle/login</code></td>
</tr>
</tbody>
</table>

The authentication path where AppRole authentication is enabled within the HashiCorp Vault server.

- **keyring_hashicorp_ca_path**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--keyring-hashicorp-ca-path=file_name</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.18</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>keyring_hashicorp_ca_path</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>File name</td>
</tr>
<tr>
<td>Default Value</td>
<td><code>empty string</code></td>
</tr>
</tbody>
</table>

The absolute path name of a local file accessible to the MySQL server that contains a properly formatted TLS certificate authority.

If this variable is not set, the `keyring_hashicorp` plugin opens an HTTPS connection without using server certificate verification, and trusts any certificate delivered by the HashiCorp Vault server. For this to be safe, it must be assumed that the Vault server is not malicious and that no man-in-the-middle attack is possible. If those assumptions are invalid, set `keyring_hashicorp_ca_path` to the path of a trusted CA certificate. (For example, for the instructions in Certificate and Key Preparation, this is the `company.crt` file.)

- **keyring_hashicorp_caching**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td>`--keyring-hashicorp-caching=[{OFF</td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.18</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>keyring_hashicorp_caching</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
</tbody>
</table>
Keyring System Variables

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>OFF</td>
</tr>
</tbody>
</table>

Whether to enable the optional in-memory key cache used by the `keyring_hashicorp` plugin to cache keys from the HashiCorp Vault server. If the cache is enabled, the plugin populates it during initialization. Otherwise, the plugin populates only the key list during initialization.

Enabling the cache is a compromise: It improves performance, but maintains a copy of sensitive key information in memory, which may be undesirable for security purposes.

- `keyring_hashicorp_commit_auth_path`

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduced</td>
<td>8.0.18</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>keyring_hashicorp_commit_auth_path</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>No</td>
</tr>
<tr>
<td><code>SET_VAR</code> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
</tbody>
</table>

This variable is associated with `keyring_hashicorp_auth_path`, from which it takes its value during `keyring_hashicorp` plugin initialization. It reflects the “committed” value actually used for plugin operation if initialization succeeds. For additional information, see `keyring_hashicorp Configuration`.

- `keyring_hashicorp_commit_ca_path`

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduced</td>
<td>8.0.18</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>keyring_hashicorp_commit_ca_path</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>No</td>
</tr>
<tr>
<td><code>SET_VAR</code> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
</tbody>
</table>

This variable is associated with `keyring_hashicorp_ca_path`. See the description of the `keyring_hashicorp_commit_auth_path` system variable for information about `_commit_` variables.

- `keyring_hashicorp_commit_caching`

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduced</td>
<td>8.0.18</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>keyring_hashicorp_commit_caching</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>No</td>
</tr>
<tr>
<td><code>SET_VAR</code> Hint Applies</td>
<td>No</td>
</tr>
</tbody>
</table>
### Keyring System Variables

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>String</td>
</tr>
</tbody>
</table>

This variable is associated with `keyring_hashicorp_caching`. See the description of the `keyring_hashicorp_commit_auth_path` system variable for information about `_commit_` variables.

- **keyring_hashicorp_commit_role_id**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Introduced</strong></td>
<td>8.0.18</td>
</tr>
<tr>
<td><strong>System Variable</strong></td>
<td><code>keyring_hashicorp_commit_role_id</code></td>
</tr>
<tr>
<td><strong>Scope</strong></td>
<td>Global</td>
</tr>
<tr>
<td><strong>Dynamic</strong></td>
<td>No</td>
</tr>
<tr>
<td><strong>SET_VAR Hint Applies</strong></td>
<td>No</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>String</td>
</tr>
</tbody>
</table>

This variable is associated with `keyring_hashicorp_role_id`. See the description of the `keyring_hashicorp_commit_auth_path` system variable for information about `_commit_` variables.

- **keyring_hashicorp_commit_server_url**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Introduced</strong></td>
<td>8.0.18</td>
</tr>
<tr>
<td><strong>System Variable</strong></td>
<td><code>keyring_hashicorp_commit_server_url</code></td>
</tr>
<tr>
<td><strong>Scope</strong></td>
<td>Global</td>
</tr>
<tr>
<td><strong>Dynamic</strong></td>
<td>No</td>
</tr>
<tr>
<td><strong>SET_VAR Hint Applies</strong></td>
<td>No</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>String</td>
</tr>
</tbody>
</table>

This variable is associated with `keyring_hashicorp_server_url`. See the description of the `keyring_hashicorp_commit_auth_path` system variable for information about `_commit_` variables.

- **keyring_hashicorp_commit_store_path**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Introduced</strong></td>
<td>8.0.18</td>
</tr>
<tr>
<td><strong>System Variable</strong></td>
<td><code>keyring_hashicorp_commit_store_path</code></td>
</tr>
<tr>
<td><strong>Scope</strong></td>
<td>Global</td>
</tr>
<tr>
<td><strong>Dynamic</strong></td>
<td>No</td>
</tr>
<tr>
<td><strong>SET_VAR Hint Applies</strong></td>
<td>No</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>String</td>
</tr>
</tbody>
</table>

This variable is associated with `keyring_hashicorp_store_path`. See the description of the `keyring_hashicorp_commit_auth_path` system variable for information about `_commit_` variables.
Keyring System Variables

- **keyring_hashicorp_role_id**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--keyring-hashicorp-role-id=value</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.18</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>keyring_hashicorp_role_id</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td><code>empty string</code></td>
</tr>
</tbody>
</table>

The HashiCorp Vault AppRole authentication role ID. The value must be in UUID format.

This variable is mandatory. If not specified, `keyring_hashicorp` initialization fails.

- **keyring_hashicorp_secret_id**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--keyring-hashicorp-secret-id=value</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.18</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>keyring_hashicorp_secret_id</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td><code>empty string</code></td>
</tr>
</tbody>
</table>

The HashiCorp Vault AppRole authentication secret ID. The value must be in UUID format.

This variable is mandatory. If not specified, `keyring_hashicorp` initialization fails.

The value of this variable is sensitive, so its value is masked by * characters when displayed.

- **keyring_hashicorp_server_url**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--keyring-hashicorp-server-url=value</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.18</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>keyring_hashicorp_server_url</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td><code>https://127.0.0.1:8200</code></td>
</tr>
</tbody>
</table>

The HashiCorp Vault server URL. The value must begin with `https://`. 
Keyring System Variables

- **keyring_hashicorp_store_path**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--keyring-hashicorp-store-path=value</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.18</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>keyring_hashicorp_store_path</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>SET_VAR</strong> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td><code>empty string</code></td>
</tr>
</tbody>
</table>

A store path within the HashiCorp Vault server that is writeable when appropriate AppRole AppRole credentials are provided by `keyring_hashicorp`. To specify the credentials, set the `keyring_hashicorp_role_id` and `keyring_hashicorp_secret_id` system variables (for example, as shown in `keyring_hashicorp Configuration`).

This variable is mandatory. If not specified, `keyring_hashicorp` initialization fails.

- **keyring_okv_conf_dir**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--keyring-okv-conf-dir=dir_name</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>keyring_okv_conf_dir</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>SET_VAR</strong> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Directory name</td>
</tr>
<tr>
<td>Default Value</td>
<td><code>empty string</code></td>
</tr>
</tbody>
</table>

The path name of the directory that stores configuration information used by the `keyring_okv` plugin. This variable is unavailable unless that plugin is installed. The location should be a directory considered for use only by the `keyring_okv` plugin. For example, do not locate the directory under the data directory.

The default `keyring_okv_conf_dir` value is empty. For the `keyring_okv` plugin to be able to access Oracle Key Vault, the value must be set to a directory that contains Oracle Key Vault configuration and SSL materials. For instructions on setting up this directory, see Section 6.4.4, “Using the `keyring_okv` KMIP Plugin”.

The directory should have a restrictive mode and be accessible only to the account used to run the MySQL server. For example, on Unix and Unix-like systems, to use the `/usr/local/mysql/mysql-keyring-okv` directory, the following commands (executed as `root`) create the directory and set its mode and ownership:

```bash
cd /usr/local/mysql
mkdir mysql-keyring-okv
chmod 750 mysql-keyring-okv
chown mysql mysql-keyring-okv
chgrp mysql mysql-keyring-okv
```

If the value assigned to `keyring_okv_conf_dir` specifies a directory that does not exist, or that does not contain configuration information that enables a connection to Oracle Key Vault to
be established, `keyring_okv` writes an error message to the error log. If an attempted runtime assignment to `keyring_okv_conf_dir` results in an error, the variable value and keyring operation remain unchanged.

- `keyring_operations`

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduced</td>
<td>8.0.4</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>keyring_operations</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><code>SET_VAR</code> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>ON</td>
</tr>
</tbody>
</table>

Whether keyring operations are enabled. This variable is used during key migration operations. See Section 6.4.7, "Migrating Keys Between Keyring Keystores". The privileges required to modify this variable are `ENCRYPTION_KEY_ADMIN` in addition to either `SYSTEM_VARIABLES_ADMIN` or `SUPER`.

### 6.5 MySQL Enterprise Audit

**Note**

MySQL Enterprise Audit is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see [https://www.mysql.com/products/](https://www.mysql.com/products/).

MySQL Enterprise Edition includes MySQL Enterprise Audit, implemented using a server plugin named `audit_log`. MySQL Enterprise Audit uses the open MySQL Audit API to enable standard, policy-based monitoring, logging, and blocking of connection and query activity executed on specific MySQL servers. Designed to meet the Oracle audit specification, MySQL Enterprise Audit provides an out of the box, easy to use auditing and compliance solution for applications that are governed by both internal and external regulatory guidelines.

When installed, the audit plugin enables MySQL Server to produce a log file containing an audit record of server activity. The log contents include when clients connect and disconnect, and what actions they perform while connected, such as which databases and tables they access.

After you install the audit plugin (see Section 6.5.2, "Installing or Uninstalling MySQL Enterprise Audit"), it writes an audit log file. By default, the file is named `audit.log` in the server data directory. To change the name of the file, set the `audit_log_file` system variable at server startup.

By default, audit log file contents are written in new-style XML format, without compression or encryption. To select the file format, set the `audit_log_format` system variable at server startup. For details on file format and contents, see Section 6.5.4, "Audit Log File Formats".

For more information about controlling how logging occurs, including audit log file naming and format selection, see Section 6.5.5, "Audit Log Logging Configuration". To perform filtering of audited events, see Section 6.5.6, "Audit Log Filtering". For descriptions of the parameters used to configure the audit log plugin, see Audit Log Options and Variables.

If the audit log plugin is enabled, the Performance Schema (see MySQL Performance Schema) has instrumentation for it. To identify the relevant instruments, use this query:

```sql
SELECT NAME FROM performance_schema.setup_instruments
WHERE NAME LIKE '%/alog/%';
```
6.5.1 Audit Log Components

MySQL Enterprise Audit is based on the audit log plugin and related components:

- A server-side plugin named `audit_log` examines auditable events and determines whether to write them to the audit log.

- User-defined functions enable manipulation of filtering definitions that control logging behavior, the encryption password, and log file reading.

- Tables in the `mysql` system database provide persistent storage of filter and user account data.

- System variables enable audit log configuration and status variables provide runtime operational information.

- An `AUDIT_ADMIN` privilege enables users to administer the audit log.

6.5.2 Installing or Uninstalling MySQL Enterprise Audit

This section describes how to install or uninstall MySQL Enterprise Audit, which is implemented using the audit log plugin and related components described in Section 6.5.1, “Audit Log Components”. For general information about installing plugins, see Installing and Uninstalling Plugins.

**Important**

Read this entire section before following its instructions. Parts of the procedure differ depending on your environment.

**Note**

If installed, the `audit_log` plugin involves some minimal overhead even when disabled. To avoid this overhead, do not install MySQL Enterprise Audit unless you plan to use it.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

To install MySQL Enterprise Audit, look in the `share` directory of your MySQL installation and choose the script that is appropriate for your platform. The available scripts differ in the suffix used to refer to the plugin library file:

- `audit_log_filter_win_install.sql`: Choose this script for Windows systems that use `.dll` as the file name suffix.

- `audit_log_filter_linux_install.sql`: Choose this script for Linux and similar systems that use `.so` as the file name suffix.

Run the script as follows. The example here uses the Linux installation script. Make the appropriate substitution for your system.

```
shell> mysql -u root -p < audit_log_filter_linux_install.sql
Enter password: (enter root password here)
```

**Note**

Some MySQL versions have introduced changes to the structure of the MySQL Enterprise Audit tables. To ensure that your tables are up to date for upgrades from earlier versions of MySQL 8.0, perform the MySQL upgrade procedure, making sure to use the option that forces an update (see Upgrading MySQL).

If you prefer to run the update statements only for the MySQL Enterprise Audit tables, see the following discussion.
As of MySQL 8.0.12, for new MySQL installations, the USER and HOST columns in the audit_log_user table used by MySQL Enterprise Audit have definitions that better correspond to the definitions of the User and Host columns in the mysql.user system table. For upgrades to an installation for which MySQL Enterprise Audit is already installed, it is recommended that you alter the table definitions as follows:

```
ALTER TABLE mysql.audit_log_user
  DROP FOREIGN KEY audit_log_user_ibfk_1;
ALTER TABLE mysql.audit_log_filter
  CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_as_ci;
ALTER TABLE mysql.audit_log_user
  CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_as_ci;
ALTER TABLE mysql.audit_log_user
  MODIFY COLUMN USER VARCHAR(32);
ALTER TABLE mysql.audit_log_user
  ADD FOREIGN KEY (FILTERNAME) REFERENCES mysql.audit_log_filter(NAME);
```

**Note**

To use MySQL Enterprise Audit in the context of master/slave replication, Group Replication, or InnoDB cluster, you must prepare the slave or secondary nodes prior to running the installation script on the master or primary node. This is necessary because the INSTALL PLUGIN statement in the script is not replicated.

1. On each slave or secondary node, extract the INSTALL PLUGIN statement from the installation script and execute it manually.
2. On the master or primary node, run the installation script as described previously.

To verify plugin installation, examine the INFORMATION_SCHEMA.PLUGINS table or use the SHOW PLUGINS statement (see Obtaining Server Plugin Information). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
  FROM INFORMATION_SCHEMA.PLUGINS
  WHERE PLUGIN_NAME LIKE 'audit%';
+-------------+---------------+
| PLUGIN_NAME | PLUGIN_STATUS |
+-------------+---------------+
| audit_log   | ACTIVE        |
+-------------+---------------+
```

If the plugin fails to initialize, check the server error log for diagnostic messages.

After MySQL Enterprise Audit is installed, you can use the --audit-log option for subsequent server startups to control audit_log plugin activation. For example, to prevent the plugin from being removed at runtime, use this option:

```
[mysqld]
audit-log=FORCE_PLUS_PERMANENT
```

If it is desired to prevent the server from running without the audit plugin, use --audit-log with a value of FORCE or FORCE_PLUS_PERMANENT to force server startup to fail if the plugin does not initialize successfully.

**Important**

By default, rule-based audit log filtering logs no auditable events for any users. This differs from legacy audit log behavior, which logs all auditable events for all users (see Section 6.5.8, “Legacy Mode Audit Log Filtering”). Should you wish to produce log-everything behavior with rule-based filtering, create a simple filter to enable logging and assign it to the default account:
6.5.3 MySQL Enterprise Audit Security Considerations

By default, contents of audit log files produced by the audit log plugin are not encrypted and may contain sensitive information, such as the text of SQL statements. For security reasons, audit log files should be written to a directory accessible only to the MySQL server and to users with a legitimate reason to view the log. The default file name is `audit.log` in the data directory. This can be changed by setting the `audit_log_file` system variable at server startup. Other audit log files may exist due to log rotation.

For additional security, enable audit log file encryption. See Audit Log File Encryption.

6.5.4 Audit Log File Formats

The MySQL server calls the audit log plugin to write an audit record to its log file whenever an auditable event occurs. Typically the first audit record written after plugin startup contains the server description and startup options. Elements following that one represent events such as client connect and disconnect events, executed SQL statements, and so forth. Only top-level statements are logged, not statements within stored programs such as triggers or stored procedures. Contents of files referenced by statements such as `LOAD DATA` are not logged.

To select the log format that the audit log plugin uses to write its log file, set the `audit_log_format` system variable at server startup. These formats are available:

- **New-style XML format** (`audit_log_format=NEW`): An XML format that has better compatibility with Oracle Audit Vault than old-style XML format. MySQL 8.0 uses new-style XML format by default.
- **Old-style XML format** (`audit_log_format=OLD`): The original audit log format used by default in older MySQL series.
- **JSON format** (`audit_log_format=JSON`)

By default, audit log file contents are written in new-style XML format, without compression or encryption.

Note

For information about issues to consider when changing the log format, see Audit Log File Format.

The following sections describe the available audit logging formats:

- **New-Style XML Audit Log File Format**
• Old-Style XML Audit Log File Format

• JSON Audit Log File Format

New-Style XML Audit Log File Format

Here is a sample log file in new-style XML format (audit_log_format=NEW), reformatted slightly for readability:

```xml
<?xml version="1.0" encoding="utf-8"?>
<AUDIT>
  <AUDIT_RECORD>
    <TIMESTAMP>2019-10-03T14:06:33 UTC</TIMESTAMP>
    <RECORD_ID>1_2019-10-03T14:06:33</RECORD_ID>
    <NAME>Audit</NAME>
    <SERVER_ID>1</SERVER_ID>
    <VERSION>1</VERSION>
    <STARTUP_OPTIONS>/usr/local/mysql/bin/mysqld
--socket=/usr/local/mysql/mysql.sock
--port=3306</STARTUP_OPTIONS>
    <OS_VERSION>i686-Linux</OS_VERSION>
    <MYSQL_VERSION>5.7.21-log</MYSQL_VERSION>
  </AUDIT_RECORD>
  <AUDIT_RECORD>
    <TIMESTAMP>2019-10-03T14:09:38 UTC</TIMESTAMP>
    <RECORD_ID>2_2019-10-03T14:06:33</RECORD_ID>
    <NAME>Connect</NAME>
    <CONNECTION_ID>5</CONNECTION_ID>
    <STATUS>0</STATUS>
    <STATUS_CODE>0</STATUS_CODE>
    <USER>root</USER>
    <OS_LOGIN/>
    <HOST>localhost</HOST>
    <IP>127.0.0.1</IP>
    <COMMAND_CLASS>connect</COMMAND_CLASS>
    <CONNECTION_TYPE>SSL/TLS</CONNECTION_TYPE>
    <CONNECTION_ATTRIBUTES>
      <ATTRIBUTE>
        <NAME>_pid</NAME>
        <VALUE>42794</VALUE>
      </ATTRIBUTE>
      ...
      <ATTRIBUTE>
        <NAME>program_name</NAME>
        <VALUE>mysqladmin</VALUE>
      </ATTRIBUTE>
    </CONNECTION_ATTRIBUTES>
    <PRIV_USER>root</PRIV_USER>
    <PROXY_USER/>
    <DB>test</DB>
  </AUDIT_RECORD>
  ...
  <AUDIT_RECORD>
    <TIMESTAMP>2019-10-03T14:09:39 UTC</TIMESTAMP>
    <RECORD_ID>8_2019-10-03T14:06:33</RECORD_ID>
    <NAME>Quit</NAME>
  </AUDIT_RECORD>
</AUDIT>
```
The audit log file is written as XML, using UTF-8 (up to 4 bytes per character). The root element is `<AUDIT>`. The root element contains `<AUDIT_RECORD>` elements, each of which provides information about an audited event. When the audit log plugin begins writing a new log file, it writes the XML declaration and opening `<AUDIT>` root element tag. When the plugin closes a log file, it writes the closing `</AUDIT>` root element tag. The closing tag is not present while the file is open.

Elements within `<AUDIT_RECORD>` elements have these characteristics:

- Some elements appear in every `<AUDIT_RECORD>` element. Others are optional and may appear depending on the audit record type.

- Order of elements within an `<AUDIT_RECORD>` element is not guaranteed.

- Element values are not fixed length. Long values may be truncated as indicated in the element descriptions given later.

- The `<`, `>`, and `&` characters are encoded as `&lt;`, `&gt;`, and `&quot;`, and `&amp;`, respectively. NUL bytes (U+00) are encoded as the `?` character.

- Characters not valid as XML characters are encoded using numeric character references. Valid XML characters are:

  #x9 | #xA | #xD | [#x20-#x7F] | [#xE000-#xFFFD] | [#x10000-#x10FFFF]

The following elements are mandatory in every `<AUDIT_RECORD>` element:

- `<NAME>`

  A string representing the type of instruction that generated the audit event, such as a command that the server received from a client.

  Example:

  `</NAME>Query</NAME>`
Audit Log File Formats

Some common `<NAME>` values:

<table>
<thead>
<tr>
<th><code>&lt;NAME&gt;</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audit</td>
<td>When auditing starts, which may be server startup time</td>
</tr>
<tr>
<td>Connect</td>
<td>When a client connects, also known as logging in</td>
</tr>
<tr>
<td>Query</td>
<td>An SQL statement (executed directly)</td>
</tr>
<tr>
<td>Prepare</td>
<td>Preparation of an SQL statement; usually followed by Execute</td>
</tr>
<tr>
<td>Execute</td>
<td>Execution of an SQL statement; usually follows Prepare</td>
</tr>
<tr>
<td>Shutdown</td>
<td>Server shutdown</td>
</tr>
<tr>
<td>Quit</td>
<td>When a client disconnects</td>
</tr>
<tr>
<td>NoAudit</td>
<td>Auditing has been turned off</td>
</tr>
</tbody>
</table>

The possible values are Audit, Binlog Dump, Change user, Close stmt, Connect Out, Connect, Create DB, Daemon, Debug, Delayed insert, Drop DB, Execute, Fetch, Field List, Init DB, Kill, Long Data, NoAudit, Ping, Prepare, Processlist, Query, Quit, Refresh, Register Slave, Reset stmt, Set option, Shutdown, Sleep, Statistics, Table Dump, TableDelete, TableInsert, TableRead, TableUpdate, Time.

Many of these values correspond to the COM_`xxx` command values listed in the my_command.h header file. For example, Create DB and Change user correspond to COM_CREATE_DB and COM_CHANGE_USER, respectively.

Events having `<NAME>` values of TableXXX accompany Query events. For example, the following statement generates one Query event, two TableRead events, and a TableInsert event:

```sql
INSERT INTO t3 SELECT t1.* FROM t1 JOIN t2;
```

Each TableXXX event contains `<DB>` and `<TABLE>` elements to identify the table to which the event refers.

- `<RECORD_ID>`

A unique identifier for the audit record. The value is composed from a sequence number and timestamp, in the format `SEQ_TIMESTAMP`. When the audit log plugin opens the audit log file, it initializes the sequence number to the size of the audit log file, then increments the sequence by 1 for each record logged. The timestamp is a UTC value in `YYYY-MM-DDTmm:ss` format indicating the date and time when the audit log plugin opened the file.

Example:

```xml
<RECORD_ID>12_2019-10-03T14:06:33</RECORD_ID>
```

- `<TIMESTAMP>`

A string representing a UTC value in `YYYY-MM-DDTmm:ss` UTC format indicating the date and time when the audit event was generated. For example, the event corresponding to execution of an SQL statement received from a client has a `<TIMESTAMP>` value occurring after the statement finishes, not when it was received.

Example:

```xml
<TIMESTAMP>2019-10-03T14:09:45 UTC</TIMESTAMP>
```

The following elements are optional in `<AUDIT_RECORD>` elements. Many of them occur only with specific `<NAME>` element values.

- `<COMMAND_CLASS>`

A string that indicates the type of action performed.

Example:

```xml
<COMMAND_CLASS>drop_table</COMMAND_CLASS>
```
The values correspond to the `statement/sql/xxx` command counters. For example, `xxx` is `drop_table` and `select` for `DROP TABLE` and `SELECT` statements, respectively. The following statement displays the possible names:

```sql
SELECT REPLACE(EVENT_NAME, 'statement/sql/', '') AS name
FROM performance_schema.events_statements_summary_global_by_event_name
WHERE EVENT_NAME LIKE 'statement/sql/%'
ORDER BY name;
```

• `<CONNECTION_ATTRIBUTES>`

As of MySQL 8.0.19, events with a `<COMMAND_CLASS>` value of `connect` may include a `<CONNECTION_ATTRIBUTES>` element to display the connection attributes passed by the client at connect time. (For information about these attributes, which are also exposed in Performance Schema tables, see Performance Schema Connection Attribute Tables.)

The `<CONNECTION_ATTRIBUTES>` element contains one `<ATTRIBUTE>` element per attribute, each of which contains `<NAME>` and `<VALUE>` elements to indicate the attribute name and value, respectively.

Example:

```
<CONNECTION_ATTRIBUTES>
  <ATTRIBUTE>
    <NAME>_pid</NAME>
    <VALUE>42794</VALUE>
  </ATTRIBUTE>
  <ATTRIBUTE>
    <NAME>_os</NAME>
    <VALUE>osx10.14</VALUE>
  </ATTRIBUTE>
  <ATTRIBUTE>
    <NAME>_platform</NAME>
    <VALUE>x86_64</VALUE>
  </ATTRIBUTE>
  <ATTRIBUTE>
    <NAME>_client_version</NAME>
    <VALUE>8.0.19</VALUE>
  </ATTRIBUTE>
  <ATTRIBUTE>
    <NAME>_client_name</NAME>
    <VALUE>libmysql</VALUE>
  </ATTRIBUTE>
  <ATTRIBUTE>
    <NAME>program_name</NAME>
    <VALUE>mysqladmin</VALUE>
  </ATTRIBUTE>
</CONNECTION_ATTRIBUTES>
```

If no connection attributes are present in the event, none are logged and no `<CONNECTION_ATTRIBUTES>` element appears. This can occur if the connection attempt is unsuccessful, the client passes no attributes, or the connection occurs internally such as during server startup or when initiated by a plugin.

• `<CONNECTION_ID>`

An unsigned integer representing the client connection identifier. This is the same as the value returned by the `CONNECTION_ID()` function within the session.

Example:

```
<CONNECTION_ID>127</CONNECTION_ID>
```
Audit Log File Formats

- **<CONNECTION_TYPE>**
  
  The security state of the connection to the server. Permitted values are TCP/IP (TCP/IP connection established without encryption), SSL/TLS (TCP/IP connection established with encryption), Socket (Unix socket file connection), Named Pipe (Windows named pipe connection), and Shared Memory (Windows shared memory connection).
  
  Example:
  
  `<CONNECTION_TYPE>SSL/TLS</CONNECTION_TYPE>`

- **<DB>**
  
  A string representing the default database name.
  
  Example:
  
  `<DB>test</DB>`

- **<HOST>**
  
  A string representing the client host name.
  
  Example:
  
  `<HOST>localhost</HOST>`

- **<IP>**
  
  A string representing the client IP address.
  
  Example:
  
  `<IP>127.0.0.1</IP>`

- **<MYSQL_VERSION>**
  
  A string representing the MySQL server version. This is the same as the value of the `VERSION()` function or `version` system variable.
  
  Example:
  
  `<MYSQL_VERSION>5.7.21-log</MYSQL_VERSION>`

- **<OS_LOGIN>**
  
  A string representing the external user name used during the authentication process, as set by the plugin used to authenticate the client. With native (built-in) MySQL authentication, or if the plugin does not set the value, this element is empty. The value is the same as that of the `external_user` system variable (see Section 4.18, “Proxy Users”).
  
  Example:
  
  `<OS_LOGIN>jeffrey</OS_LOGIN>`

- **<OS_VERSION>**
  
  A string representing the operating system on which the server was built or is running.
  
  Example:
  
  `<OS_VERSION>x86_64-Linux</OS_VERSION>`
Audit Log File Formats

- **<PRIV_USER>**
  A string representing the user that the server authenticated the client as. This is the user name that the server uses for privilege checking, and may differ from the `<USER>` value.
  
  Example:
  
  `<PRIV_USER>jeffrey</PRIV_USER>`

- **<PROXY_USER>**
  A string representing the proxy user (see Section 4.18, “Proxy Users”). The value is empty if user proxying is not in effect.

  Example:
  
  `<PROXY_USER>developer</PROXY_USER>`

- **<SERVER_ID>**
  An unsigned integer representing the server ID. This is the same as the value of the `server_id` system variable.

  Example:
  
  `<SERVER_ID>1</SERVER_ID>`

- **<SQLTEXT>**
  A string representing the text of an SQL statement. The value can be empty. Long values may be truncated. The string, like the audit log file itself, is written using UTF-8 (up to 4 bytes per character), so the value may be the result of conversion. For example, the original statement might have been received from the client as an SJIS string.

  Example:
  
  `<SQLTEXT>DELETE FROM t1</SQLTEXT>`

- **<STARTUP_OPTIONS>**
  A string representing the options that were given on the command line or in option files when the MySQL server was started. The first option is the path to the server executable.

  Example:
  
  `<STARTUP_OPTIONS>/usr/local/mysql/bin/mysqld --port=3306 --log_output=FILE</STARTUP_OPTIONS>`

- **<STATUS>**
  An unsigned integer representing the command status: 0 for success, nonzero if an error occurred. This is the same as the value of the `mysql_errno()` C API function. See the description for `<STATUS_CODE>` for information about how it differs from `<STATUS>`.

  The audit log does not contain the SQLSTATE value or error message. To see the associations between error codes, SQLSTATE values, and messages, see Server Error Message Reference.

  Warnings are not logged.

  Example:
  
  `<STATUS>1051</STATUS>`

- **<STATUS_CODE>**
Audit Log File Formats

An unsigned integer representing the command status: 0 for success, 1 if an error occurred.

The STATUS_CODE value differs from the STATUS value: STATUS_CODE is 0 for success and 1 for error, which is compatible with the EZ_collector consumer for Audit Vault. STATUS is the value of the mysql_errno() C API function. This is 0 for success and nonzero for error, and thus is not necessarily 1 for error.

Example:

```
<STATUS_CODE>0</STATUS_CODE>
```

- `<TABLE>`

A string representing a table name.

Example:

```
<TABLE>t3</TABLE>
```

- `<USER>`

A string representing the user name sent by the client. This may differ from the `<PRIV_USER>` value.

Example:

```
<USER>root[root] @ localhost [127.0.0.1]</USER>
```

- `<VERSION>`

An unsigned integer representing the version of the audit log file format.

Example:

```
<VERSION>1</VERSION>
```

Old-Style XML Audit Log File Format

Here is a sample log file in old-style XML format (audit_log_format=OLD), reformatted slightly for readability:

```
<?xml version="1.0" encoding="utf-8"?>
<AUDIT>
  <AUDIT_RECORD
    TIMESTAMP="2019-10-03T14:25:00 UTC"
    RECORD_ID="1_2019-10-03T14:25:00"
    NAME="Audit"
    SERVER_ID="1"
    VERSION="1"
    STARTUP_OPTIONS="--port=3306"
    OS_VERSION="i686-Linux"
    MYSQL_VERSION="5.7.21-log"/>
  <AUDIT_RECORD
    TIMESTAMP="2019-10-03T14:25:24 UTC"
    RECORD_ID="2_2019-10-03T14:25:00"
    NAME="Connect"
    CONNECTION_ID="4"
    STATUS="0"
    STATUS_CODE="0"
    USER="root"
    OS_LOGIN=""
    HOST="localhost"
    IP="127.0.0.1"
    COMMAND_CLASS="connect"
    CONNECTION_TYPE="SSL/TLS"
    PRIV_USER="root"
    PROXY_USER=""
```

281
The audit log file is written as XML, using UTF-8 (up to 4 bytes per character). The root element is `<AUDIT>`. The root element contains `<AUDIT_RECORD>` elements, each of which provides information about an audited event. When the audit log plugin begins writing a new log file, it writes the XML declaration and opening `<AUDIT>` root element tag. When the plugin closes a log file, it writes the closing `</AUDIT>` root element tag. The closing tag is not present while the file is open.

Attributes of `<AUDIT_RECORD>` elements have these characteristics:

- Some attributes appear in every `<AUDIT_RECORD>` element. Others are optional and may appear depending on the audit record type.
- Order of attributes within an `<AUDIT_RECORD>` element is not guaranteed.
- Attribute values are not fixed length. Long values may be truncated as indicated in the attribute descriptions given later.
- The `<`, `>`, `,`, and `&` characters are encoded as `&lt;`, `&gt;`, `&quot;`, and `&amp;`, respectively. NUL bytes (U+00) are encoded as the `?` character.
- Characters not valid as XML characters are encoded using numeric character references. Valid XML characters are:

  ```plaintext
  #x9 | #xA | #xD | [#x20-#xD7FF] | [#xE000-#xFFFD] | [#x10000-#x10FFFF]
  ```

The following attributes are mandatory in every `<AUDIT_RECORD>` element:

- **NAME**

  A string representing the type of instruction that generated the audit event, such as a command that the server received from a client.

  **Example:** `NAME="Query"`
Some common NAME values:

<table>
<thead>
<tr>
<th>NAME</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audit</td>
<td>When auditing starts, which may be server startup time</td>
</tr>
<tr>
<td>Connect</td>
<td>When a client connects, also known as logging in</td>
</tr>
<tr>
<td>Query</td>
<td>An SQL statement (executed directly)</td>
</tr>
<tr>
<td>Prepare</td>
<td>Preparation of an SQL statement; usually followed by Execute</td>
</tr>
<tr>
<td>Execute</td>
<td>Execution of an SQL statement; usually follows Prepare</td>
</tr>
<tr>
<td>Shutdown</td>
<td>Server shutdown</td>
</tr>
<tr>
<td>Quit</td>
<td>When a client disconnects</td>
</tr>
<tr>
<td>NoAudit</td>
<td>Auditing has been turned off</td>
</tr>
</tbody>
</table>

The possible values are Audit, Binlog Dump, Change user, Close stmt, Connect Out, Connect, Create DB, Daemon, Debug, Delayed insert, Drop DB, Execute, Fetch, Field List, Init DB, Kill, Long Data, NoAudit, Ping, Prepare, Processlist, Query, Quit, Refresh, Register Slave, Reset stmt, Set option, Shutdown, Sleep, Statistics, Table Dump, TableDelete, TableInsert, TableRead, TableUpdate, Time.

Many of these values correspond to the COM_xxx command values listed in the my_command.h header file. For example, "Create DB" and "Change user" correspond to COM_CREATE_DB and COM_CHANGE_USER, respectively.

Events having NAME values of TableXXX accompany Query events. For example, the following statement generates one Query event, two TableRead events, and a TableInsert event:

```sql
INSERT INTO t3 SELECT t1.* FROM t1 JOIN t2;
```

Each TableXXX event has DB and TABLE attributes to identify the table to which the event refers.

Connect events for old-style XML audit log format do not include connection attributes.

- **RECORD_ID**

  A unique identifier for the audit record. The value is composed from a sequence number and timestamp, in the format SEQ_TIMESTAMP. When the audit log plugin opens the audit log file, it initializes the sequence number to the size of the audit log file, then increments the sequence by 1 for each record logged. The timestamp is a UTC value in YYYY-MM-DDThh:mm:ss format indicating the date and time when the audit log plugin opened the file.

  Example: RECORD_ID="12_2019-10-03T14:25:00"

- **TIMESTAMP**

  A string representing a UTC value in YYYY-MM-DDThh:mm:ss UTC format indicating the date and time when the audit event was generated. For example, the event corresponding to execution of an SQL statement received from a client has a TIMESTAMP value occurring after the statement finishes, not when it was received.

  Example: TIMESTAMP="2019-10-03T14:25:32 UTC"

The following attributes are optional in <AUDIT_RECORD> elements. Many of them occur only for elements with specific values of the NAME attribute.

- **COMMAND_CLASS**

  A string that indicates the type of action performed.

  Example: COMMAND_CLASS="drop_table"

  The values correspond to the statement/sql/xxx command counters. For example, xxx is drop_table and select for DROP TABLE and SELECT statements, respectively. The following statement displays the possible names:

  ```sql
  SELECT REPLACE(EVENT_NAME, 'statement/sql/', '') AS name
  ```
Audit Log File Formats

```
FROM performance_schema.events_statements_summary_global_by_event_name
WHERE EVENT_NAME LIKE 'statement/sql/%'
ORDER BY name;
```

- **CONNECTION_ID**
  
  An unsigned integer representing the client connection identifier. This is the same as the value returned by the `CONNECTION_ID()` function within the session.

  Example: `CONNECTION_ID="127"`

- **CONNECTION_TYPE**
  
  The security state of the connection to the server. Permitted values are **TCP/IP** (TCP/IP connection established without encryption), **SSL/TLS** (TCP/IP connection established with encryption), **Socket** (Unix socket file connection), **Named Pipe** (Windows named pipe connection), and **Shared Memory** (Windows shared memory connection).

  Example: `CONNECTION_TYPE="SSL/TLS"`

- **DB**
  
  A string representing the default database name.

  Example: `DB="test"`

- **HOST**
  
  A string representing the client host name.

  Example: `HOST="localhost"`

- **IP**
  
  A string representing the client IP address.

  Example: `IP="127.0.0.1"`

- **MYSQL_VERSION**
  
  A string representing the MySQL server version. This is the same as the value of the `VERSION()` function or `version` system variable.

  Example: `MYSQL_VERSION="5.7.21-log"`

- **OS_LOGIN**
  
  A string representing the external user name used during the authentication process, as set by the plugin used to authenticate the client. With native (built-in) MySQL authentication, or if the plugin does not set the value, this attribute is empty. The value is the same as that of the `external_user` system variable (see Section 4.18, "Proxy Users").

  Example: `OS_LOGIN="jeffrey"`

- **OS_VERSION**
  
  A string representing the operating system on which the server was built or is running.

  Example: `OS_VERSION="x86_64-Linux"`

- **PRIV_USER**
  
  A string representing the user that the server authenticated the client as. This is the user name that the server uses for privilege checking, and it may differ from the `USER` value.
Audit Log File Formats

Example: PRIV_USER="jeffrey"

• PROXY_USER

A string representing the proxy user (see Section 4.18, “Proxy Users”). The value is empty if user proxying is not in effect.

Example: PROXY_USER="developer"

• SERVER_ID

An unsigned integer representing the server ID. This is the same as the value of the server_id system variable.

Example: SERVER_ID="1"

• SQLTEXT

A string representing the text of an SQL statement. The value can be empty. Long values may be truncated. The string, like the audit log file itself, is written using UTF-8 (up to 4 bytes per character), so the value may be the result of conversion. For example, the original statement might have been received from the client as an SJIS string.

Example: SQLTEXT="DELETE FROM t1"

• STARTUP_OPTIONS

A string representing the options that were given on the command line or in option files when the MySQL server was started.

Example: STARTUP_OPTIONS="--port=3306 --log_output=FILE"

• STATUS

An unsigned integer representing the command status: 0 for success, nonzero if an error occurred. This is the same as the value of the mysql_errno() C API function. See the description for STATUS_CODE for information about how it differs from STATUS.

The audit log does not contain the SQLSTATE value or error message. To see the associations between error codes, SQLSTATE values, and messages, see Server Error Message Reference.

Warnings are not logged.

Example: STATUS="1051"

• STATUS_CODE

An unsigned integer representing the command status: 0 for success, 1 if an error occurred.

The STATUS_CODE value differs from the STATUS value: STATUS_CODE is 0 for success and 1 for error, which is compatible with the EZ_collector consumer for Audit Vault. STATUS is the value of the mysql_errno() C API function. This is 0 for success and nonzero for error, and thus is not necessarily 1 for error.

Example: STATUS_CODE="0"

• TABLE

A string representing a table name.

Example: TABLE="t3"
Audit Log File Formats

• **USER**
  A string representing the user name sent by the client. This may differ from the PRIV_USER value.

• **VERSION**
  An unsigned integer representing the version of the audit log file format.

  Example: VERSION="1"

### JSON Audit Log File Format

For JSON-format audit logging (audit_log_format=JSON), the log file contents form a JSON array with each array element representing an audited event as a JSON hash of key-value pairs. Examples of complete event records appear later in this section. The following is an excerpt of partial events:

```
[ {
  "timestamp": "2019-10-03 13:50:01",
  "id": 0,
  "class": "audit",
  "event": "startup",
  ...
},
  {
    "timestamp": "2019-10-03 15:02:32",
    "id": 0,
    "class": "connection",
    "event": "connect",
    ...
  },
  ...
  {
    "timestamp": "2019-10-03 17:37:26",
    "id": 0,
    "class": "table_access",
    "event": "insert",
    ...
  }
]
```

The audit log file is written using UTF-8 (up to 4 bytes per character). When the audit log plugin begins writing a new log file, it writes the opening [ array marker. When the plugin closes a log file, it writes the closing ] array marker. The closing marker is not present while the file is open.

Items within audit records have these characteristics:

• Some items appear in every audit record. Others are optional and may appear depending on the audit record type.

• Order of items within an audit record is not guaranteed.

• Item values are not fixed length. Long values may be truncated as indicated in the item descriptions given later.

• The " and \ characters are encoded as \" and \\, respectively.

The following examples show the JSON object formats for different event types (as indicated by the class and event items), reformatted slightly for readability:

Auditing startup event:

```
{ "timestamp": "2019-10-03 14:21:56",
  "id": 0,
  "class": "audit",
  "event": "startup",
  "connection_id": 0,
  "startup_data": { "server_id": 1,
```
When the audit log plugin starts as a result of server startup (as opposed to being enabled at runtime), `connection_id` is set to 0, and `account` and `login` are not present.

**Auditing shutdown event:**

```json
{
  "timestamp": "2019-10-03 14:28:20",
  "id": 3,
  "class": "audit",
  "event": "shutdown",
  "connection_id": 0,
  "shutdown_data": { "server_id": 1 }
}
```

When the audit log plugin is uninstalled as a result of server shutdown (as opposed to being disabled at runtime), `connection_id` is set to 0, and `account` and `login` are not present.

**Connect or change-user event:**

```json
{
  "timestamp": "2019-10-03 14:23:18",
  "id": 1,
  "class": "connection",
  "event": "connect",
  "connection_id": 5,
  "account": { "user": "root", "host": "localhost" },
  "login": { "user": "root", "os": "", "ip": "::1", "proxy": "" },
  "connection_data": { "connection_type": "ssl",
    "status": 0,
    "db": "test",
    "connection_attributes": {
      ".pid": "43236",
      ...
      "program_name": "mysqladmin"
    }
}
```

**Disconnect event:**

```json
{
  "timestamp": "2019-10-03 14:24:45",
  "id": 3,
  "class": "connection",
  "event": "disconnect",
  "connection_id": 5,
  "account": { "user": "root", "host": "localhost" },
  "login": { "user": "root", "os": "", "ip": "::1", "proxy": "" },
  "connection_data": { "connection_type": "ssl" }
}
```

**Query event:**

```json
{
  "timestamp": "2019-10-03 14:23:35",
  "id": 2,
  "class": "general",
  "event": "status",
  "connection_id": 5,
  "account": { "user": "root", "host": "localhost" },
  "login": { "user": "root", "os": "", "ip": "::1", "proxy": "" },
  "general_data": { "command": "Query",
    "sql_command": "show_variables",
    "query": "SHOW VARIABLES",
    "status": 0 }
}
```

**Table access event (read, delete, insert, update):**

```json
{
  "timestamp": "2019-10-03 14:23:41",
  "id": 0,
  "class": "table_access",
```
The items in the following list appear at the top level of JSON-format audit records: Each item value is either a scalar or a JSON hash. For items that have a hash value, the description lists only the item names within that hash. For more complete descriptions of second-level hash items, see later in this section.

- **account**

  The MySQL account associated with the event. The value is a hash containing these items equivalent to the value of the `CURRENT_USER()` function within the section: user, host.

  Example:

  ```json
  "account": { "user": "root", "host": "localhost" }
  ``

- **class**

  A string representing the event class. The class defines the type of event, when taken together with the `event` item that specifies the event subclass.

  Example:

  ```json
  "class": "connection"
  ``

  The following table shows the permitted combinations of **class** and **event** values.

  **Table 6.17 Audit Log Class and Event Combinations**

<table>
<thead>
<tr>
<th>Class Value</th>
<th>Permitted Event Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit</td>
<td>startup, shutdown</td>
</tr>
<tr>
<td>connection</td>
<td>connect, change_user, disconnect</td>
</tr>
<tr>
<td>general</td>
<td>status</td>
</tr>
<tr>
<td>table_access_data</td>
<td>read, delete, insert, update</td>
</tr>
</tbody>
</table>

- **connection_data**

  Information about a client connection. The value is a hash containing these items: `connection_type`, `status`, `db`, and possibly `connection_attributes`. This item occurs only for audit records with a **class** value of `connection`.

  Example:

  ```json
  "connection_data": { "connection_type": "ssl", "status": 0, "db": "test" }
  ``

  As of MySQL 8.0.19, events with a **class** value of `connection` and **event** value of `connect` may include a `connection_attributes` item to display the connection attributes passed by the client at connect time. (For information about these attributes, which are also exposed in Performance Schema tables, see Performance Schema Connection Attribute Tables.)

  The `connection_attributes` value is a hash that represents each attribute by its name and value.
Audit Log File Formats

Example:

```
"connection_attributes": {
    "_pid": "43236",
    "_os": "osx10.14",
    "_platform": "x86_64",
    "_client_version": "8.0.19",
    "_client_name": "libmysql",
    "program_name": "mysqladmin"
}
```

If no connection attributes are present in the event, none are logged and no `connection_attributes` item appears. This can occur if the connection attempt is unsuccessful, the client passes no attributes, or the connection occurs internally such as during server startup or when initiated by a plugin.

- **connection_id**

  An unsigned integer representing the client connection identifier. This is the same as the value returned by the `CONNECTION_ID()` function within the session.

  Example:

  ```json
  "connection_id": 5
  ```

- **event**

  A string representing the subclass of the event class. The subclass defines the type of event, when taken together with the `class` item that specifies the event class. For more information, see the `class` item description.

  Example:

  ```json
  "event": "connect"
  ```

- **general_data**

  Information about an executed statement or command. The value is a hash containing these items: `command`, `sql_command`, `query`, `status`. This item occurs only for audit records with a `class` value of `general`.

  Example:

  ```json
  "general_data": { "command": "Query",
                   "sql_command": "show_variables",
                   "query": "SHOW VARIABLES",
                   "status": 0 }
  ```

- **id**

  An unsigned integer representing an event ID.

  Example:

  ```json
  "id": 2
  ```

  For audit records that have the same `timestamp` value, their `id` values distinguish them and form a sequence. Within the audit log, `timestamp/id` pairs are unique. These pairs are bookmarks that identify event locations within the log.

- **login**

  Information indicating how a client connected to the server. The value is a hash containing these items: `user, os, ip, proxy`. 


Audit Log File Formats

Example:

```json
"login": { "user": "root", "os": "", "ip": "::1", "proxy": "" }
```

- **shutdown_data**

Information pertaining to audit log plugin termination. The value is a hash containing these items:

- **server_id**: This item occurs only for audit records with **class** and **event** values of **audit** and **shutdown**, respectively.

Example:

```json
"shutdown_data": { "server_id": 1 }
```

- **startup_data**

Information pertaining to audit log plugin initialization. The value is a hash containing these items:

- **server_id**, **os_version**, **mysql_version**, **args**. This item occurs only for audit records with **class** and **event** values of **audit** and **startup**, respectively.

Example:

```json
"startup_data": { "server_id": 1,
  "os_version": "i686-Linux",
  "mysql_version": "5.7.21-log",
  "args": ["/usr/local/mysql/bin/mysqld",
            "--loose-audit-log-format=JSON",
            "--log-error=log.err",
            "--pid-file=mysqld.pid",
            "--port=3306" ] }
```

- **table_access_data**

Information about an access to a table. The value is a hash containing these items: **db**, **table**, **query**, **sql_command**. This item occurs only for audit records with a **class** value of **table_access**.

Example:

```json
"table_access_data": { "db": "test",
  "table": "t1",
  "query": "INSERT INTO t1 (i) VALUES(1),(2),(3)",
  "sql_command": "insert" }
```

- **timestamp**

A string representing a UTC value in **YYYY-MM-DD hh:mm:ss** format indicating the date and time when the audit event was generated. For example, the event corresponding to execution of an SQL statement received from a client has a **timestamp** value occurring after the statement finishes, not when it was received.

Example:

```json
"timestamp": "2019-10-03 13:50:01"
```

For audit records that have the same **timestamp** value, their **id** values distinguish them and form a sequence. Within the audit log, **timestamp/id** pairs are unique. These pairs are bookmarks that identify event locations within the log.

These items appear within hash values associated with top-level items of JSON-format audit records:

- **args**
Audit Log File Formats

An array of options that were given on the command line or in option files when the MySQL server was started. The first option is the path to the server executable.

Example:

```
"args": ["/usr/local/mysql/bin/mysqld",
    "--loose-audit-log-format=JSON",
    "--log-error=log.err",
    "--pid-file=mysqld.pid",
    "--port=3306"
]
```

- **command**

A string representing the type of instruction that generated the audit event, such as a command that the server received from a client.

Example:

```
"command": "Query"
```

- **connection_type**

The security state of the connection to the server. Permitted values are tcp/ip (TCP/IP connection established without encryption), ssl (TCP/IP connection established with encryption), socket (Unix socket file connection), named_pipe (Windows named pipe connection), and shared_memory (Windows shared memory connection).

Example:

```
"connection_type": "tcp/tcp"
```

- **db**

A string representing a database name. For connection_data, it is the default database. For table_access_data, it is the table database.

Example:

```
"db": "test"
```

- **host**

A string representing the client host name.

Example:

```
"host": "localhost"
```

- **ip**

A string representing the client IP address.

Example:

```
"ip": "::1"
```

- **mysql_version**

A string representing the MySQL server version. This is the same as the value of the VERSION() function or version system variable.

Example:

```
"mysql_version": "5.7.21-log"
```
Audit Log File Formats

- **os**

  A string representing the external user name used during the authentication process, as set by the plugin used to authenticate the client. With native (built-in) MySQL authentication, or if the plugin does not set the value, this attribute is empty. The value is the same as that of the `external_user` system variable. See Section 4.18, “Proxy Users”.

  Example:

  ```json
  "os": "jeffrey"
  ```

- **os_version**

  A string representing the operating system on which the server was built or is running.

  Example:

  ```json
  "os_version": "i686-Linux"
  ```

- **proxy**

  A string representing the proxy user (see Section 4.18, “Proxy Users”). The value is empty if user proxying is not in effect.

  Example:

  ```json
  "proxy": "developer"
  ```

- **query**

  A string representing the text of an SQL statement. The value can be empty. Long values may be truncated. The string, like the audit log file itself, is written using UTF-8 (up to 4 bytes per character), so the value may be the result of conversion. For example, the original statement might have been received from the client as an SJIS string.

  Example:

  ```json
  "query": "DELETE FROM t1"
  ```

- **server_id**

  An unsigned integer representing the server ID. This is the same as the value of the `server_id` system variable.

  Example:

  ```json
  "server_id": 1
  ```

- **sql_command**

  A string that indicates the SQL statement type.

  Example:

  ```json
  "sql_command": "insert"
  ```

  The values correspond to the `statement/sql/xxx` command counters. For example, `xxx` is `drop_table` and `select` for `DROP_TABLE` and `SELECT` statements, respectively. The following statement displays the possible names:

  ```sql
  SELECT REPLACE(EVENT_NAME, 'statement/sql/', '') AS name
  FROM performance_schema.events_statements_summary_global_by_event_name
  WHERE EVENT_NAME LIKE 'statement/sql/%'
  ORDER BY name;
  ```
• **status**

An unsigned integer representing the command status: 0 for success, nonzero if an error occurred. This is the same as the value of the `mysql_errno()` C API function.

The audit log does not contain the SQLSTATE value or error message. To see the associations between error codes, SQLSTATE values, and messages, see [Server Error Message Reference](#).

Warnings are not logged.

Example:

```
"status": 1051
```

• **table**

A string representing a table name.

Example:

```
"table": "t1"
```

• **user**

A string representing a user name. The meaning differs depending on the item within which user occurs:

- Within account items, user is a string representing the user that the server authenticated the client as. This is the user name that the server uses for privilege checking.

- Within login items, user is a string representing the user name sent by the client.

Example:

```
"user": "root"
```

### 6.5.5 Audit Log Logging Configuration

This section describes how to configure audit logging characteristics, such as the file to which the audit log plugin writes events, the format of written events, and whether to enable log file compression and encryption.

- **Audit Log File Name**
- **Audit Log File Format**
- **Audit Log File Compression**
- **Audit Log File Encryption**
- **Audit Log File Manual Uncompression and Decryption**
- **Audit Log File Encryption Prior to MySQL 8.0.17**
- **Audit Log File Space Management and Name Rotation**
- **Audit Logging Write Strategy**
- **Audit Log File Reading**

**Note**

Encryption capabilities described here apply as of MySQL 8.0.17, with the exception of the section that compares current encryption capabilities to the
Audit Log Logging Configuration

previous more-limited capabilities (see Audit Log File Encryption Prior to MySQL 8.0.17).

For additional information about the user-defined functions and system variables that affect audit logging, see Audit Log Functions, and Audit Log Options and Variables.

The audit log plugin can also control which audited events are written to the audit log file, based on event content or the account from which events originate. See Section 6.5.6, “Audit Log Filtering”.

Audit Log File Name

To configure the audit log file name, set the audit_log_file system variable at server startup. By default, the name is audit.log in the server data directory. For security reasons, write the audit log file to a directory accessible only to the MySQL server and to users with a legitimate reason to view the log.

The plugin interprets the audit_log_file value as composed of a base name and an optional suffix. If compression or encryption are enabled, the effective file name (the name actually used to create the log file) differs from the configured file name because it has additional suffixes:

• If compression is enabled, the plugin adds a suffix of .gz.
• If encryption is enabled, the plugin adds a suffix of .pwd_id.enc, where pwd_id indicates which encryption password to use for log file operations.

The effective audit log file name is the name resulting from the addition of applicable compression and encryption suffixes to the configured file name. For example, if the configured audit_log_file value is audit.log, the effective file name is one of the values shown in the following table.

<table>
<thead>
<tr>
<th>Enabled Features</th>
<th>Effective File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>No compression or encryption</td>
<td>audit.log</td>
</tr>
<tr>
<td>Compression</td>
<td>audit.log.gz</td>
</tr>
<tr>
<td>Encryption</td>
<td>audit.log.pwd_id.enc</td>
</tr>
<tr>
<td>Compression, encryption</td>
<td>audit.log.gz.pwd_id.enc</td>
</tr>
</tbody>
</table>

pwd_id indicates the ID of the password used to encrypt or decrypt a file. pwd_id format is pwd_timestamp-seq, where:

• pwd_timestamp is a UTC value in YYYYMMDDThhmmss format indicating when the password was created.
• seq is a sequence number. Sequence numbers start at 1 and increase for passwords that have the same pwd_timestamp value.

Here are some example password ID values:

<table>
<thead>
<tr>
<th>Password ID Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>20190403T142359-1</td>
</tr>
<tr>
<td>20190403T142400-1</td>
</tr>
<tr>
<td>20190403T142400-2</td>
</tr>
</tbody>
</table>

The audit log plugin stores encryption passwords in the keyring (see Section 6.4, “The MySQL Keyring”). The IDs of audit log passwords in the keyring are based on pwd_id values, with a prefix of audit_log-. For the example password IDs just shown, the corresponding keyring IDs are:

<table>
<thead>
<tr>
<th>Keyring Password IDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit_log-20190403T142359-1</td>
</tr>
<tr>
<td>audit_log-20190403T142400-1</td>
</tr>
<tr>
<td>audit_log-20190403T142400-2</td>
</tr>
</tbody>
</table>

The password currently used for encryption by the audit log plugin is the one having the largest pwd_timestamp value. If multiple passwords have the largest pwd_timestamp value, the current password is the one with the largest sequence number. For example, in the preceding set of password
IDs, two of them have the largest timestamp, 20190403T142400, so the current password is the one with the largest sequence number (2).

The audit log plugin performs certain actions during initialization and termination based on the effective audit log file name:

- During initialization, the plugin checks whether a file with the audit log file name already exists and renames it if so. (In this case, the plugin assumes that the previous server invocation exited unexpectedly with the audit log plugin running.) The plugin then writes to a new empty audit log file.
- During termination, the plugin renames the audit log file.
- File renaming (whether during plugin initialization or termination) occurs according to the usual rules for automatic log file rotation; see Automatic Audit Log File Rotation.

Audit Log File Format

To configure the audit log file format, set the audit_log_format system variable at server startup. By default, the format is NEW (new-style XML format). For details about each format, see Section 6.5.4, “Audit Log File Formats”.

If you change audit_log_format, it is recommended that you also change audit_log_file. Otherwise, there will be two sets of log files with the same base name but different formats.

Audit Log File Compression

Audit log file compression can be enabled for any logging format.

To configure audit log file compression, set the audit_log_compression system variable at server startup. Permitted values are NONE (no compression; the default) and GZIP (GNU Zip compression).

If both compression and encryption are enabled, compression occurs before encryption. To recover the original file manually, first decrypt it, then uncompress it. See Audit Log File Manual Uncompression and Decryption.

Audit Log File Encryption

Audit log file encryption can be enabled for any logging format. Encryption is based on user-defined passwords (with the exception of the initial password that the audit log plugin generates). To use this feature, the MySQL keyring must be enabled because audit logging uses it for password storage. Any keyring plugin can be used; for instructions, see Section 6.4, “The MySQL Keyring”.

To configure audit log file encryption, set the audit_log_encryption system variable at server startup. Permitted values are NONE (no encryption; the default) and AES (AES-256-CBC cipher encryption).

To set or get an encryption password, use these user-defined functions (UDFs):

- To set the current encryption password, invoke audit_log_encryption_password_set(). This function stores the new password in the keyring, If encryption is enabled, it also performs a log file rotation operation that renames the current log file, and begins a new log file encrypted with the password. File renaming occurs according to the usual rules for automatic log file rotation; see Automatic Audit Log File Rotation.
- To get the current encryption password, invoke audit_log_encryption_password_get() with no argument. To get a password by ID, pass an argument specifying the keyring ID of the current password or an archived password.

To determine which audit log keyring IDs exist, query the Performance Schema keyring_keys table:

```sql
mysql> SELECT * FROM performance_schema.keyring_keys
```
Audit Log Logging Configuration

```
WHERE KEY_ID LIKE 'audit_log%'
ORDER BY KEY_ID;
```

<table>
<thead>
<tr>
<th>KEY_ID</th>
<th>KEY_OWNER</th>
<th>BACKEND_KEY_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit_log-20190415T152248-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_log-20190415T153507-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_log-20190416T125122-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_log-20190416T141608-1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For additional information about audit log encryption functions, see Audit Log Functions.

When the audit log plugin initializes, if it finds that log file encryption is enabled, it checks whether the keyring contains an audit log encryption password. If not, the plugin automatically generates a random initial encryption password and stores it in the keyring. To discover this password, invoke `audit_log_encryption_password_get()`.

If both compression and encryption are enabled, compression occurs before encryption. To recover the original file manually, first decrypt it, then uncompress it. See Audit Log File Manual Uncompression and Decryption.

### Audit Log File Manual Uncompression and Decryption

Audit log files can be uncompressed and decrypted using standard tools. This should be done only for log files that have been closed (archived) and are no longer in use, not for the log file that the audit log plugin is currently writing. You can recognize archived log files because they have been renamed by the audit log plugin to include a timestamp in the file name just after the base name.

For this discussion, assume that `audit_log_file` is set to `audit.log`. In that case, an archived audit log file has one of the names shown in the following table.

<table>
<thead>
<tr>
<th>Enabled Features</th>
<th>Archived File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>No compression or encryption</td>
<td><code>audit.timestamp.log</code></td>
</tr>
<tr>
<td>Compression</td>
<td><code>audit.timestamp.log.gz</code></td>
</tr>
<tr>
<td>Encryption</td>
<td><code>audit.timestamp.log.pwd_id.enc</code></td>
</tr>
<tr>
<td>Compression, encryption</td>
<td><code>audit.timestamp.log.gz.pwd_id.enc</code></td>
</tr>
</tbody>
</table>

As discussed in Audit Log File Name, `pwd_id` format is `pwd_timestamp-seq`. Thus, the names of archived encrypted log files actually contain two timestamps. The first indicates rotation time, and the second indicates when the password was created.

Consider the following set of archived encrypted log file names:

```
audit.20190410T205827.log.20190403T185337-1.enc
audit.20190410T210243.log.20190403T185337-1.enc
audit.20190415T145309.log.20190414T223342-1.enc
audit.20190415T151322.log.20190414T223342-2.enc
```

Each file name has a unique rotation-time timestamp. By contrast, the password timestamps are not unique:

- The first two files have the same password ID and sequence number (`20190403T185337-1`). They have the same encryption password.
- The second two files have the same password ID (`20190414T223342`) but different sequence numbers (1, 2). These files have different encryption passwords.

To uncompress a compressed log file manually, use `gzip, gunzip -d`, or equivalent command. For example:

```
gunzip -c audit.timestamp.log.gz > audit.timestamp.log
```
To decrypt an encrypted log file manually, use the `openssl` command. For example:

```
openssl enc -d -aes-256-cbc -pass pass:password -md sha256 -in audit.timestamp.log.pwd_id.enc -out audit.timestamp.log
```

If both compression and encryption are enabled for audit logging, compression occurs before encryption. In this case, the file name has `.gz` and `.pwd_id.enc` suffixes added, corresponding to the order in which those operations occur. To recover the original file manually, perform the operations in reverse. That is, first decrypt the file, then uncompress it:

```
openssl enc -d -aes-256-cbc -pass pass:password -md sha256 -in audit.timestamp.log.gz.pwd_id.enc -out audit.timestamp.log.gz

gunzip -c audit.timestamp.log.gz > audit.timestamp.log
```

### Audit Log File Encryption Prior to MySQL 8.0.17

This section covers the differences in audit log file encryption capabilities prior to and as of MySQL 8.0.17, which is when password archiving and expiration were implemented. It also indicates how the audit log plugin handles upgrades to MySQL 8.0.17 or higher from versions lower than 8.0.17.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Prior to MySQL 8.0.17</th>
<th>As of MySQL 8.0.17</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of passwords</td>
<td>Single password only</td>
<td>Multiple passwords permitted</td>
</tr>
<tr>
<td>Encrypted log file names</td>
<td>.enc suffix</td>
<td>.pwd_id.enc suffix</td>
</tr>
<tr>
<td>Password keyring ID</td>
<td>audit_log</td>
<td>audit_log-pwd_id</td>
</tr>
<tr>
<td>Password history</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Password expiration</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Prior to MySQL 8.0.17, there is no password history, so setting a new password makes the old password inaccessible, rendering MySQL Enterprise Audit unable to read log files encrypted with the old password. You must keep a record of the previous password should you need to decrypt those files manually.

If audit log file encryption is enabled when you upgrade to MySQL 8.0.17 or higher from a lower version, the audit log plugin performs these upgrade actions:

- During plugin initialization, the plugin checks for an encryption password with a keyring ID of `audit_log`. If it finds one, the plugin duplicates the password using a keyring ID in `audit_log-pwd_id` format and uses it as the current encryption password.

- Existing encrypted log files have a suffix of `.enc`. The plugin does not rename these to have a suffix of `.pwd_id.enc`, but can read them as long as the key with the ID of `audit_log` remains in the keyring.

- When password cleanup occurs, if the plugin expires any password with a keyring ID in `audit_log-pwd_id` format, it also expires the password with a keyring ID of `audit_log`, if it exists. (At this point, encrypted log files that have a suffix of `.enc` rather than `.pwd_id.enc` become unreadable by the plugin, so it is assumed that you no longer need them.)

### Audit Log File Space Management and Name Rotation

The audit log file has the potential to grow very large and consume a lot of disk space. To enable management of the space used by its log files, the audit log plugin provides for log file rotation, either manually or automatically. Rotation capabilities use the `audit_log_flush` and `audit_log_rotate_on_size` system variables:

- By default, `audit_log_rotate_on_size=0` and no log rotation occurs unless performed manually. In this case, use `audit_log_flush` to close and reopen the current log file after manually renaming it.
• If `audit_log_rotate_on_size` is greater than 0, automatic rotation occurs when a write to the current log file causes its size to exceed this value. The audit log plugin closes the file, renames it, and opens a new log file. With automatic rotation enabled, `audit_log_flush` has no effect.

• Automatic rotation also occurs under several other conditions, described later.

```
Note
Renamed log files are not removed automatically. For example, with size-based log file rotation, renamed log files do not rotate off the end of the name sequence. Instead, they have unique names and accumulate indefinitely. To avoid excessive space use, remove old files periodically, backing them up first as necessary. If backed-up log files are encrypted, also back up the corresponding encryption passwords to a safe place, should you need to decrypt the files later.
```

The following discussion describes log file rotation methods in greater detail.

• Manual Audit Log File Rotation

• Automatic Audit Log File Rotation

**Manual Audit Log File Rotation**

If `audit_log_rotate_on_size=0` (the default), no log rotation occurs unless performed manually. In this case, the audit log plugin closes and reopens the log file when the `audit_log_flush` value changes from disabled to enabled. Log file renaming must be done externally to the server. Suppose that the log file name is `audit.log` and you want to maintain the three most recent log files, cycling through the names `audit.log.1` through `audit.log.3`. On Unix, perform rotation manually like this:

1. From the command line, rename the current log files:
   
   ```
   mv audit.log.2 audit.log.3
   mv audit.log.1 audit.log.2
   mv audit.log audit.log.1
   ```
   
   This strategy overwrites the current `audit.log.3` contents, placing a bound on the number of archived log files and the space they use.

2. At this point, the plugin is still writing to the current log file, which has been renamed to `audit.log.1`. Connect to the server and flush the log file so the plugin closes it and reopens a new `audit.log` file:
   
   ```
   SET GLOBAL audit_log_flush = ON;
   ```
   
   `audit_log_flush` is special in that its value remains `OFF` so that you need not disable it explicitly before enabling it again to perform another flush.

```
Note
If compression or encryption are enabled, log file names include suffixes that signify the enabled features, as well as a password ID if encryption is enabled. If file names include a password ID, be sure to retain the ID in the name of any files you rename manually so that the password to use for decryption operations can be determined.
```

```
Note
For JSON-format logging, renaming audit log files manually makes them unavailable to the log-reading functions because the audit log plugin no longer can determine that they are part of the log file sequence (see Audit Log File Reading). Consider setting `audit_log_rotate_on_size` greater than 0 to use size-based rotation instead.
```
Automatic Audit Log File Rotation

If `audit_log_rotate_on_size` is greater than 0, setting `audit_log_flush` has no effect. Instead, whenever a write to the current log file causes its size to exceed the `audit_log_rotate_on_size` value, the audit log plugin closes the file, renames it, and opens a new log file.

Automatic rotation also occurs under these conditions:

- During plugin initialization, if a file with the audit log file name already exists (see Audit Log File Name).
- During plugin termination.
- When the `audit_log_encryption_password_set()` function is called to set the encryption password, if encryption is enabled. (Rotation does not occur if encryption is disabled.)

The plugin renames the original file by inserting a timestamp just after its base name. For example, if the file name is `audit.log`, the plugin renames it to a value such as `audit.20190115T140633.log`. The timestamp is a UTC value in `YYYYMMDDThhmmss` format. The timestamp indicates rotation time for XML logging, and the timestamp of the last event written to the file for JSON logging.

Audit Logging Write Strategy

The audit log plugin can use any of several strategies for log writes. Regardless of strategy, logging occurs on a best-effort basis, with no guarantee of consistency.

To specify a write strategy, set the `audit_log_strategy` system variable at server startup. By default, the strategy value is `ASYNCHRONOUS` and the plugin logs asynchronously to a buffer, waiting if the buffer is full. It’s possible to tell the plugin not to wait (`PERFORMANCE`) or to log synchronously, either using file system caching (`SEMISYNCHRONOUS`) or forcing output with a `sync()` call after each write request (`SYNCHRONOUS`).

For asynchronous write strategy, the `audit_log_buffer_size` system variable is the buffer size in bytes. Set this variable at server startup to change the buffer size. The plugin uses a single buffer, which it allocates when it initializes and removes when it terminates. The plugin does not allocate this buffer for nonasynchronous write strategies.

Asynchronous logging strategy has these characteristics:

- Minimal impact on server performance and scalability.
- Blocking of threads that generate audit events for the shortest possible time; that is, time to allocate the buffer plus time to copy the event to the buffer.
- Output goes to the buffer. A separate thread handles writes from the buffer to the log file.

With asynchronous logging, the integrity of the log file may be compromised if a problem occurs during a write to the file or if the plugin does not shut down cleanly (for example, in the event that the server host exits unexpectedly). To reduce this risk, set `audit_log_strategy` to use synchronous logging.

A disadvantage of `PERFORMANCE` strategy is that it drops events when the buffer is full. For a heavily loaded server, the audit log may have events missing.

Audit Log File Reading

The audit log plugin enables bookmarking and reading of JSON-format audit log files. (These capabilities do not apply to files written in other log formats.)

When the audit log plugin initializes and is configured for JSON logging, it uses the directory containing the audit log file (determined from the `audit_log_file` value) as the location to search for readable
Audit Log Logging Configuration

Audit log files. To do this, it uses the value of `audit_log_file` to determine the file base name and suffix values, then looks for files with names that match the following pattern, where `[...]` indicates optional file name parts:

```
basename.[timestamp].suffix[.gz][.pwd_id].enc
```

A file is encrypted if its name ends with `.enc`. The plugin determines the keyring ID of the decryption password needed to read it as follows:

- If `.enc` is preceded by `pwd_id`, the keyring ID is `audit_log-pwd_id`.
- If `.enc` is not preceded by `pwd_id`, the file has an old name from before password history was implemented. The keyring ID is `audit_log`.

The plugin opens each file matching the pattern, checks that the file really contains JSON audit records, and sorts the files using the timestamps from the first record of each file to construct a list of files that are subject to use with the log-reading functions.

The plugin cannot include in the sequence files that were renamed manually and do not match the preceding pattern, or that were encrypted with a password no longer available in the keyring.

To read events from the audit log, use these user-defined functions (UDFs):

- `audit_log_read_bookmark()` returns a JSON string representing a bookmark for the most recently written audit log event. This bookmark is suitable for passing to `audit_log_read()` to indicate to that function where to begin reading. Example bookmark:
  ```json
  { "timestamp": "2019-10-03 21:03:44", "id": 0 }
  ```

- `audit_log_read()` reads events from the audit log and returns a JSON string containing an array of audit events.

Example `audit_log_read()` invocation using the current bookmark:

```sql
mysql> SELECT audit_log_read(audit_log_read_bookmark());
+-----------------------------------------------------------------------+
| audit_log_read(audit_log_read_bookmark())                             |
+-----------------------------------------------------------------------+
| [ { "timestamp": "2019-10-03 22:41:24", "id": 0, "class": "connection", ... } |
+-----------------------------------------------------------------------+
```

Each event in the `audit_log_read()` return value is a JSON hash, except that the last array element may be a JSON null value to indicate no following events are available to read. For example:

```json
[
  { "timestamp": "2019-10-03 22:08:08", "id": 10,
    "class": "general", "event": "status",
    ... }
  { "timestamp": "2019-10-03 22:08:08", "id": 11,
    "class": "connection", "event": "disconnect",
    ... }
  { "timestamp": "2019-10-03 13:39:33", "id": 0,
    "class": "connection", "event": "connect",
    ... }
  { "timestamp": "2019-10-03 13:39:33", "id": 1,
    "class": "general", "event": "status",
    ... }
  { "timestamp": "2019-10-03 13:39:33", "id": 2,
    "class": "connection", "event": "disconnect",
    ...
```
Use `audit_log_read()` like this:

- For the first call to `audit_log_read()` within a session, pass a bookmark indicating where to begin reading.
- If the final value of the returned array is not a JSON null value, there are more events following those just read and `audit_log_read()` can be called without or with a bookmark argument. Without an argument, reading continues with the next unread event. With a bookmark argument, reading continues from the bookmark.
- If the final value of the returned array is a JSON null value, there are no more events left to be read and the next call to `audit_log_read()` must include a bookmark argument.

A bookmark is a JSON hash that indicates where and how much to read. The following items are significant in the bookmark value (other items are ignored):

- `timestamp`, `id`: The location within the audit log of the first event to read. Both items must be present to completely specify a position.
- `max_array_length`: The maximum number of events to read from the log. If omitted, the default is to read to the end of the log or until the read buffer is full, whichever comes first.

The result returned from either log-reading function is a string, which can be manipulated as necessary. Suppose that a bookmark has this value:

```sql
mysql> SET @mark := audit_log_read_bookmark();
mysql> SELECT @mark;
+-------------------------------------------------------------------+
| @mark                                                             |
+-------------------------------------------------------------------+
| { "timestamp": "2019-10-03 16:10:28", "id": 2 }                 |
+-------------------------------------------------------------------+
```

Calling `audit_log_read()` with that bookmark can return multiple events. To limit `audit_log_read()` to reading a single event, add to the bookmark a `max_array_length` item with a value of 1. For example, convert the preceding bookmark as follows:

```sql
mysql> SET @mark := JSON_SET(@mark, '$.max_array_length', 1);
mysql> SELECT @mark;
+---------------------------------------------------------------------------------+
| @mark                                                                          |
+---------------------------------------------------------------------------------+
| {"id": 2, "timestamp": "2019-10-03 16:10:28", "max_array_length": 1}         |
+---------------------------------------------------------------------------------+
```

The modified bookmark, when passed to `audit_log_read()`, produces a result of a single audit record.

Prior to MySQL 8.0.19, string return values from audit log UDFs are binary strings. To use a binary string with functions that require a nonbinary string (such as functions that manipulate JSON values), perform a conversion to `utf8mb4`. For example, before passing the bookmark to `JSON_SET()`, convert it as follows:

```sql
SET @mark = CONVERT(@mark USING utf8mb4);
```

That statement can be included even for MySQL 8.0.19 and higher because for those versions it is essentially a no-op and is harmless.

To set a limit on the number of bytes that `audit_log_read()` reads, set the `audit_log_read_buffer_size` system variable. As of MySQL 8.0.12, this variable has
Audit Log Filtering

A default of 32KB and can be set at runtime. Each client should set its session value of `audit_log_read_buffer_size` appropriately for its use of `audit_log_read()`. Prior to MySQL 8.0.12, `audit_log_read_buffer_size` has a default of 1MB, affects all clients, and can be changed only at server startup.

Each call to `audit_log_read()` returns as many available items as fit within the buffer size, skipping items that do not fit within the buffer size. Given this behavior, consider these factors when assessing the proper buffer size for an application:

- There is a tradeoff between number of calls to `audit_log_read()` and items returned per call. With a smaller buffer size, calls return fewer items, so more calls are needed. With a larger buffer size, calls return more items, so fewer calls are needed.
- With a smaller buffer size, such as the default size of 32KB, there is a greater chance that items will exceed the buffer size and `audit_log_read()` will skip them. Skipped items generate warnings.

For additional information about audit log-reading functions, see Audit Log Functions.

6.5.6 Audit Log Filtering

This section describes how audit log filtering works if the audit log plugin and the accompanying audit tables and UDFs are installed. If the plugin is installed but not the accompanying audit tables and UDFs, the plugin operates in legacy filtering mode, described in Section 6.5.8, “Legacy Mode Audit Log Filtering”. Legacy mode is filtering behavior as it was prior to MySQL 5.7.13; that is, before the introduction of rule-based filtering.

The audit log plugin has the capability of controlling logging of audited events by filtering them:

- Audited events can be filtered using these characteristics:
  - User account
  - Audit event class
  - Audit event subclass
  - Value of event fields such as those that indicate operation status or SQL statement executed

- Audit filtering is rule based:
  - A filter definition creates a set of auditing rules. Definitions can be configured to include or exclude events for logging based on the characteristics just described.
  - Filter rules have the capability of blocking (aborting) execution of qualifying events, in addition to existing capabilities for event logging.
  - Multiple filters can be defined, and any given filter can be assigned to any number of user accounts.
  - It is possible to define a default filter to use with any user account that has no explicitly assigned filter.
  - Audit filters can be defined, displayed, and modified using an SQL interface based on user-defined functions (UDFs).
  - Audit filter definitions are stored in the tables in the `mysql` system database.
  - Within a given session, the value of the read-only `audit_log_filter_id` system variable indicates whether a filter has been assigned to the session.
Note

By default, rule-based audit log filtering logs no auditable events for any users. To log all auditable events for all users, use the following statements, which create a simple filter to enable logging and assign it to the default account:

```sql
SELECT audit_log_filter_set_filter('log_all', '{ "filter": { "log": true } }');
SELECT audit_log_filter_set_user('%', 'log_all');
```

The filter assigned to `%' is used for connections from any account that has no explicitly assigned filter (which initially is true for all accounts).

The following list briefly summarizes the UDFs that implement the SQL interface for audit filtering control:

- `audit_log_filter_set_filter()`: Define a filter
- `audit_log_filter_remove_filter()`: Remove a filter
- `audit_log_filter_set_user()`: Start filtering a user account
- `audit_log_filter_remove_user()`: Stop filtering a user account
- `audit_log_filter_flush()`: Flush manual changes to the filter tables to affect ongoing filtering

For usage examples and complete details about the filtering functions, see `Using Audit Log Filtering Functions`, and `Audit Log Functions`.

Audit log filtering functions are subject to these constraints:

- To use any filtering function, the `audit_log` plugin must be enabled or an error occurs. In addition, the audit tables must exist or an error occurs. To install the `audit_log` plugin and its accompanying UDFs and tables, see `Section 6.5.2, "Installing or Uninstalling MySQL Enterprise Audit"`.

- To use any filtering function, a user must possess the `SUPER` privilege or an error occurs. To grant the `SUPER` privilege to a user account, use this statement:

  ```sql
  GRANT SUPER ON *.* TO user;
  ```

  Alternatively, should you prefer to avoid granting the `SUPER` privilege while still permitting users to access specific filtering functions, “wrapper” stored programs can be defined. This technique is described in the context of keyring UDFs in `Using General-Purpose Keyring Functions`; it can be adapted for use with filtering UDFs.

- The `audit_log` plugin operates in legacy mode if it is installed but the accompanying audit tables and functions are not created. The plugin writes these messages to the error log at server startup:

  ```
  [Warning] Plugin audit_log reported: 'Failed to open the audit log filter tables.'
  [Warning] Plugin audit_log reported: 'Audit Log plugin supports a filtering, which has not been installed yet. Audit Log plugin will run in the legacy mode, which will be disabled in the next release.'
  ```

  In legacy mode, filtering can be done based only on event account or status. For details, see `Section 6.5.8, “Legacy Mode Audit Log Filtering”`.

- `Using Audit Log Filtering Functions`

Using Audit Log Filtering Functions

Before using the audit log user-defined functions (UDFs), install them according to the instructions provided in `Section 6.5.2, "Installing or Uninstalling MySQL Enterprise Audit"`. The `SUPER` privilege is required to use any of these functions.

The audit log filtering functions enable filtering control by providing an interface to create, modify, and remove filter definitions and assign filters to user accounts.
Filter definitions are JSON values. For information about using JSON data in MySQL, see The JSON Data Type. This section shows some simple filter definitions. For more information about filter definitions, see Section 6.5.7, "Writing Audit Log Filter Definitions".

When a connection arrives, the audit log plugin determines which filter to use for the new session by searching for the user account name in the current filter assignments:

- If a filter is assigned to the user, the audit log uses that filter.
- Otherwise, if no user-specific filter assignment exists, but there is a filter assigned to the default account (%), the audit log uses the default filter.
- Otherwise, the audit log selects no audit events from the session for processing.

If a change-user operation occurs during a session (see mysql_change_user()), filter assignment for the session is updated using the same rules but for the new user.

By default, no accounts have a filter assigned, so no processing of auditable events occurs for any account.

Suppose that instead you want the default to be to log only connection-related activity (for example, to see connect, change-user, and disconnect events, but not the SQL statements users execute while connected). To achieve this, define a filter (shown here named log_conn_events) that enables logging only of events in the connection class, and assign that filter to the default account, represented by the % account name:

```sql
SET @f = '{ "filter": { "class": { "name": "connection" } } }';
SELECT audit_log_filter_set_filter('log_conn_events', @f);
SELECT audit_log_filter_set_user('%', 'log_conn_events');
```

Now the audit log uses this default account filter for connections from any account that has no explicitly defined filter.

To assign a filter explicitly to a particular user account or accounts, define the filter, then assign it to the relevant accounts:

```sql
SELECT audit_log_filter_set_filter('log_all', '{ "filter": { "log": true } }');
SELECT audit_log_filter_set_user('user1@localhost', 'log_all');
SELECT audit_log_filter_set_user('user2@localhost', 'log_all');
```

Now full logging is enabled for user1@localhost and user2@localhost. Connections from other accounts continue to be filtered using the default account filter.

To disassociate a user account from its current filter, either unassign the filter or assign a different filter:

- To unassign the filter from the user account:

  ```sql
  SELECT audit_log_filter_remove_user('user1@localhost');
  ```

  Filtering of current sessions for the account remains unaffected. Subsequent connections from the account are filtered using the default account filter if there is one, and are not logged otherwise.

- To assign a different filter to the user account:

  ```sql
  SELECT audit_log_filter_set_filter('log_nothing', '{ "filter": { "log": false } }');
  SELECT audit_log_filter_set_user('user1@localhost', 'log_nothing');
  ```

  Filtering of current sessions for the account remains unaffected. Subsequent connections from the account are filtered using the new filter. For the filter shown here, that means no logging for new connections from user1@localhost.

For audit log filtering, user name and host name comparisons are case-sensitive. This differs from comparisons for privilege checking, for which host name comparisons are not case-sensitive.

To remove a filter, do this:
Removing a filter also unassigns it from any users to whom it has been assigned, including any current sessions for those users.

The filtering UDFs just described affect audit filtering immediately and update the audit log tables in the `mysql` system database that store filters and user accounts (see Audit Log Tables). It is also possible to modify the audit log tables directly using statements such as `INSERT`, `UPDATE`, and `DELETE`, but such changes do not affect filtering immediately. To flush your changes and make them operational, call `audit_log_filter_flush()`:

```sql
SELECT audit_log_filter_flush();
```

**Warning**

`audit_log_filter_flush()` should be used only after modifying the audit tables directly, to force reloading all filters. Otherwise, this function should be avoided. It is, in effect, a simplified version of unloading and reloading the `audit_log` plugin with `UNINSTALL PLUGIN` plus `INSTALL PLUGIN`.

`audit_log_filter_flush()` affects all current sessions and detaches them from their previous filters. Current sessions are no longer logged unless they disconnect and reconnect, or execute a change-user operation.

To determine whether a filter has been assigned to the current session, check the session value of the read-only `audit_log_filter_id` system variable. If the value is 0, no filter is assigned. A nonzero value indicates the internally maintained ID of the assigned filter:

```sql
mysql> SELECT @@audit_log_filter_id;
+-----------------------+
| @@audit_log_filter_id |
|                     2 |
+-----------------------+
```

---

### 6.5.7 Writing Audit Log Filter Definitions

Filter definitions are JSON values. For information about using JSON data in MySQL, see The JSON Data Type.

Filter definitions have this form, where `actions` indicates how filtering takes place:

```json
{ "filter": actions }
```

The following discussion describes permitted constructs in filter definitions.

- Logging All Events
- Logging Specific Event Classes
- Logging Specific Event Subclasses
- Inclusive and Exclusive Logging
- Testing Event Field Values
- Blocking Execution of Specific Events
- Logical Operators
- Referencing Predefined Variables
- Referencing Predefined Functions
- Replacing a User Filter
Writing Audit Log Filter Definitions

Logging All Events

To explicitly enable or disable logging of all events, use a `log` element in the filter:

```json
{  
  "filter": {  
    "log": true
  }
}
```

The `log` value can be either `true` or `false`.

The preceding filter enables logging of all events. It is equivalent to:

```json
{  
  "filter": {}
}
```

Logging behavior depends on the `log` value and whether `class` or `event` items are specified:

- With `log` specified, its given value is used.
- Without `log` specified, logging is `true` if no `class` or `event` item is specified, and `false` otherwise (in which case, `class` or `event` can include their own `log` item).

Logging Specific Event Classes

To log events of a specific class, use a `class` element in the filter, with its `name` field denoting the name of the class to log:

```json
{  
  "filter": {  
    "class": {  
      "name": "connection"
    }
  }
}
```

The `name` value can be `connection`, `general`, or `table_access` to log connection, general, or table-access events, respectively.

The preceding filter enables logging of events in the `connection` class. It is equivalent to the following filter with `log` items made explicit:

```json
{  
  "filter": {  
    "log": false,  
    "class": {  
      "log": true,  
      "name": "connection"
    }
  }
}
```

To enable logging of multiple classes, define the `class` value as a JSON array element that names the classes:

```json
{  
  "filter": {  
    "class": [  
      {  
        "name": "connection"
      },  
      {  
        "name": "general"
      },  
      {  
        "name": "table_access"
      }
    ]
  }
}
```

Note

When multiple instances of a given item appear at the same level within a filter definition, the item values can be combined into a single instance of that item within an array value. The preceding definition can be written like this:

```json
{  
  "filter": {  
    "class": [  
      {  
        "name": "connection"
      }
    ]
  }
}
```
Logging Specific Event Subclasses

To select specific event subclasses, use an event item containing a name item that names the subclasses. The default action for events selected by an event item is to log them. For example, this filter enables logging for the named event subclasses:

```
{
  "filter": {
    "class": [
      {
        "name": "connection",
        "event": [
          { "name": "connect" },
          { "name": "disconnect" }
        ]
      },
      { "name": "general" },
      { "name": "table_access",
        "event": [
          { "name": "insert" },
          { "name": "delete" },
          { "name": "update" }
        ]
      }
    ]
  }
}
```

The event item can also contain explicit log items to indicate whether to log qualifying events. This event item selects multiple events and explicitly indicates logging behavior for them:

```
"event": [
  { "name": "read", "log": false },
  { "name": "insert", "log": true },
  { "name": "delete", "log": true },
  { "name": "update", "log": true }
]
```

The event item can also indicate whether to block qualifying events, if it contains an abort item. For details, see Blocking Execution of Specific Events.

Table 6.18, “Event Class and Subclass Combinations” describes the permitted subclass values for each event class.

**Table 6.18 Event Class and Subclass Combinations**

<table>
<thead>
<tr>
<th>Event Class</th>
<th>Event Subclass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connection</td>
<td>connect</td>
<td>Connection initiation (successful or unsuccessful)</td>
</tr>
<tr>
<td>connection</td>
<td>change_user</td>
<td>User re-authentication with different user/password during session</td>
</tr>
<tr>
<td>connection</td>
<td>disconnect</td>
<td>Connection termination</td>
</tr>
<tr>
<td>general</td>
<td>status</td>
<td>General operation information</td>
</tr>
<tr>
<td>message</td>
<td>internal</td>
<td>Internally generated message</td>
</tr>
<tr>
<td>message</td>
<td>user</td>
<td>Message generated by <code>audit_api_message_emit_udf()</code></td>
</tr>
<tr>
<td>table_access</td>
<td>read</td>
<td>Table read statements, such as <code>SELECT</code> or <code>INSERT INTO ... SELECT</code></td>
</tr>
</tbody>
</table>
Writing Audit Log Filter Definitions

<table>
<thead>
<tr>
<th>Event Class</th>
<th>Event Subclass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_access</td>
<td>delete</td>
<td>Table delete statements, such as DELETE or TRUNCATE TABLE</td>
</tr>
<tr>
<td>table_access</td>
<td>insert</td>
<td>Table insert statements, such as INSERT or REPLACE</td>
</tr>
<tr>
<td>table_access</td>
<td>update</td>
<td>Table update statements, such as UPDATE</td>
</tr>
</tbody>
</table>

Table 6.19, “Log and Abort Characteristics Per Event Class and Subclass Combination” describes for each event subclass whether it can be logged or aborted.

### Table 6.19 Log and Abort Characteristics Per Event Class and Subclass Combination

<table>
<thead>
<tr>
<th>Event Class</th>
<th>Event Subclass</th>
<th>Can be Logged</th>
<th>Can be Aborted</th>
</tr>
</thead>
<tbody>
<tr>
<td>connection</td>
<td>connect</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>connection</td>
<td>change_user</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>connection</td>
<td>disconnect</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>general</td>
<td>status</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>message</td>
<td>internal</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>message</td>
<td>user</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>table_access</td>
<td>read</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>table_access</td>
<td>delete</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>table_access</td>
<td>insert</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>table_access</td>
<td>update</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Inclusive and Exclusive Logging

A filter can be defined in inclusive or exclusive mode:

- Inclusive mode logs only explicitly specified items.
- Exclusive mode logs everything but explicitly specified items.

To perform inclusive logging, disable logging globally and enable logging for specific classes. This filter logs `connect` and `disconnect` events in the `connection` class, and events in the `general` class:

```json
{
  "filter": {
    "log": false,
    "class": {
      "name": "connection",
      "event": [
        { "name": "connect", "log": true },
        { "name": "disconnect", "log": true }
      ],
      { "name": "general", "log": true }
    }
  }
}
```

To perform exclusive logging, enable logging globally and disable logging for specific classes. This filter logs everything except events in the `general` class:

```json
{
  "filter": {
    "log": true,
    "class": {
      "name": "general", "log": false
    }
  }
}
```
This filter logs `change_user` events in the `connection` class, `message` events, and `table_access` events:

```
{
  "filter": {
    "log": true,
    "class": [
      {
        "name": "connection",
        "event": [
          { "name": "connect", "log": false },
          { "name": "disconnect", "log": false }
        ]
      },
      { "name": "general", "log": false }
    ]
  }
}
```

**Testing Event Field Values**

To enable logging based on specific event field values, specify a `field` item within the `log` item that indicates the field name and its expected value:

```
{
  "filter": {
    "class": {
      "name": "general",
      "event": {
        "name": "status",
        "log": {
          "field": { "name": "general_command.str", "value": "Query" }
        }
      }
    }
  }
}
```

Each event contains event class-specific fields that can be accessed from within a filter to perform custom filtering.

A connection event indicates when a connection-related activity occurs during a session, such as a user connecting to or disconnecting from the server. Table 6.20, "Connection Event Fields" indicates the permitted fields for connection events.

**Table 6.20 Connection Event Fields**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>integer</td>
<td>Event status:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: OK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Otherwise: Failed</td>
</tr>
<tr>
<td>connection_id</td>
<td>unsigned int</td>
<td>Connection ID</td>
</tr>
<tr>
<td>user.str</td>
<td>string</td>
<td>User name specified during authentication</td>
</tr>
<tr>
<td>user.length</td>
<td>unsigned int</td>
<td>User name length</td>
</tr>
<tr>
<td>priv_user.str</td>
<td>string</td>
<td>Authenticated user name (account user name)</td>
</tr>
<tr>
<td>priv_user.length</td>
<td>unsigned int</td>
<td>Authenticated user name length</td>
</tr>
<tr>
<td>external_user.str</td>
<td>string</td>
<td>External user name (provided by third-party authentication plugin)</td>
</tr>
<tr>
<td>external_user.length</td>
<td>unsigned int</td>
<td>External user name length</td>
</tr>
</tbody>
</table>
### Field Name | Field Type | Description
---|---|---
`proxy_user.str` | string | Proxy user name
`proxy_user.length` | unsigned integer | Proxy user name length
`host.str` | string | Connected user host
`host.length` | unsigned integer | Connected user host length
`ip.str` | string | Connected user IP address
`ip.length` | unsigned integer | Connected user IP address length
`database.str` | string | Database name specified at connect time
`database.length` | unsigned integer | Database name length
`connection_type` | integer | Connection type:
  - `::undefined`: Undefined
  - `::tcp/ip`: TCP/IP
  - `::socket`: Socket
  - `::named_pipe`: Named pipe
  - `::ssl`: TCP/IP with encryption
  - `::shared_memory`: Shared memory

The `::xxx` values are symbolic pseudo-constants that may be given instead of the literal numeric values. They must be quoted as strings and are case-sensitive.

A general event indicates the status code of an operation and its details. Table 6.21, “General Event Fields” indicates the permitted fields for general events.

### Table 6.21 General Event Fields

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
</table>
`general_error_code` | integer | Event status:
  - 0: OK
  - Otherwise: Failed
`general_thread_id` | unsigned integer | Connection/thread ID
`general_user.str` | string | User name specified during authentication
`general_user.length` | unsigned integer | User name length
`general_command.str` | string | Command name
`general_command.length` | unsigned integer | Command name length
`general_query.str` | string | SQL statement text
`general_query.length` | unsigned integer | SQL statement text length
`general_host.str` | string | Host name
`general_host.length` | unsigned integer | Host name length
`general_sql_command.str` | string | SQL command type name
`general_sql_command.length` | unsigned integer | SQL command type name length
`general_external_user.str` | string | External user name (provided by third-party authentication plugin)
`general_external_user.length` | unsigned integer | External user name length
Writing Audit Log Filter Definitions

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>general_ip.str</td>
<td>string</td>
<td>Connected user IP address</td>
</tr>
<tr>
<td>general_ip.length</td>
<td>unsigned integer</td>
<td>Connection user IP address length</td>
</tr>
</tbody>
</table>

**general_command.str** indicates a command name: Query, Execute, Quit, or Change user.

A general event with the **general_command.str** field set to Query or Execute contains **general_sql_command.str** set to a value that specifies the type of SQL command: alter_db, alter_db_upgrade, admin_commands, and so forth. These values can be seen as the last components of the Performance Schema instruments displayed by this statement:

```sql
mysql> SELECT NAME FROM performance_schema.setup_instruments WHERE NAME LIKE 'statement/sql/%' ORDER BY NAME;
+---------------------------------------+
| NAME                                  |
+---------------------------------------+
| statement/sql/alter_db                |
| statement/sql/alter_db_upgrade        |
| statement/sql/alter_event             |
| statement/sql/alter_function          |
| statement/sql/alter_instance          |
| statement/sql/alter_procedure         |
| statement/sql/alter_server            |
| ...                                   |
```

A table-access event provides information about specific table accesses. **Table 6.22, “Table-Access Event Fields”** indicates the permitted fields for table-access events.

**Table 6.22 Table-Access Event Fields**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connection_id</td>
<td>unsigned integer</td>
<td>Event connection ID</td>
</tr>
<tr>
<td>sql_command_id</td>
<td>integer</td>
<td>SQL command ID</td>
</tr>
<tr>
<td>query.str</td>
<td>string</td>
<td>SQL statement text</td>
</tr>
<tr>
<td>query.length</td>
<td>unsigned integer</td>
<td>SQL statement text length</td>
</tr>
<tr>
<td>table_database.str</td>
<td>string</td>
<td>Database name associated with event</td>
</tr>
<tr>
<td>table_database.length</td>
<td>unsigned integer</td>
<td>Database name length</td>
</tr>
<tr>
<td>table_name.str</td>
<td>string</td>
<td>Table name associated with event</td>
</tr>
<tr>
<td>table_name.length</td>
<td>unsigned integer</td>
<td>Table name length</td>
</tr>
</tbody>
</table>

The following list shows which statements produce which table-access events:

- **read event:**
  - SELECT
  - INSERT ... SELECT (for tables referenced in SELECT clause)
  - REPLACE ... SELECT (for tables referenced in SELECT clause)
  - UPDATE ... WHERE (for tables referenced in WHERE clause)
  - HANDLER ... READ

- **delete event:**
  - DELETE
  - TRUNCATE TABLE
• **insert** event:
  - INSERT
  - INSERT ... SELECT (for table referenced in INSERT clause)
  - REPLACE
  - REPLACE ... SELECT (for table referenced in REPLACE clause)
  - LOAD DATA
  - LOAD XML

• **update** event:
  - UPDATE
  - UPDATE ... WHERE (for tables referenced in UPDATE clause)

### Blocking Execution of Specific Events

**event** items can include an **abort** item that indicates whether to prevent qualifying events from executing. For example, **abort** enables rules to be written that block execution of specific SQL statements.

The **abort** item must appear within an **event** item. For example:

```json
"event": {
  "name": qualifying event subclass names
  "abort": condition
}
```

For event subclasses selected by the **name** item, the **abort** action is true or false, depending on **condition** evaluation. If the condition evaluates to true, the event is blocked. Otherwise, the event continues executing.

The **condition** specification can be as simple as **true** or **false**, or it can be more complex such that evaluation depends on event characteristics.

This filter blocks **INSERT**, **UPDATE**, and **DELETE** statements:

```json
{
  "filter": {
    "class": {
      "name": "table_access",
      "event": {
        "name": [ "insert", "update", "delete" ],
        "abort": true
      }
    }
  }
}
```

This more complex filter blocks the same statements, but only for a specific table **(finances.bank_account)**:

```json
{
  "filter": {
    "class": {
      "name": "table_access",
      "event": {
        "name": [ "insert", "update", "delete" ],
        "abort": {
          "and": {
            "field": { "name": "table_database.str", "value": "finances" } },
            "field": { "name": "table_name.str", "value": "bank_account" } }
      }
    }
  }
}
```
Statements matched and blocked by the filter return an error to the client:

ERROR 1045 (28000): Statement was aborted by an audit log filter

Not all events can be blocked (see Table 6.19, “Log and Abort Characteristics Per Event Class and Subclass Combination”). For an event that cannot, the audit log writes a warning to the error log rather than blocking it.

For attempts to define a filter in which the abort item appears elsewhere than in an event item, an error occurs.

**Logical Operators**

Logical operators (and, or, not) can be used in log items. This permits construction of more advanced filtering configurations:

```json
{  
"filter": {  
  "class": {  
    "name": "general",  
    "event": {  
      "name": "status",  
      "log": {  
        "or": [  
          {  
            "and": [  
              { "field": { "name": "general_command.str",  "value": "Query" } },  
              { "field": { "name": "general_command.length",  "value": 5 } }  
            ]  
          },  
          {  
            "and": [  
              { "field": { "name": "general_command.str",  "value": "Execute" } },  
              { "field": { "name": "general_command.length",  "value": 7 } }  
            ]  
          ]  
        ]  
      }  
    }  
  }  
}  
}
```

**Referencing Predefined Variables**

To refer to a predefined variable in a log condition, use a variable item, which tests equality against a given value:

```json
{  
"filter": {  
  "class": {  
    "name": "general",  
    "event": {  
      "name": "status",  
      "log": {  
        "variable": {  
          "name": "audit_log_connection_policy_value",  "value": "::none"  
        }  
      }  
    }  
  }  
}  
}
```
Each predefined variable corresponds to a system variable. By writing a filter that tests a predefined variable, you can modify filter operation by setting the corresponding system variable, without having to redefine the filter. For example, by writing a filter that tests the value of the `audit_log_connection_policy_value` predefined variable, you can modify filter operation by changing the value of the `audit_log_connection_policy` system variable.

The `audit_log_xxx_policy` system variables are used for the legacy mode audit log (see Section 6.5.8, "Legacy Mode Audit Log Filtering"). With rule-based audit log filtering, those variables remain visible (for example, using `SHOW VARIABLES`), but changes to them have no effect unless you write filters containing constructs that refer to them.

The following list describes the permitted predefined variables for `variable` items:

- **`audit_log_connection_policy_value`**

  This variable corresponds to the value of the `audit_log_connection_policy` system variable. The value is an unsigned integer. Table 6.23, "`audit_log_connection_policy_value` Values" shows the permitted values and the corresponding `audit_log_connection_policy` values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Corresponding audit_log_connection_policy Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 or &quot;::none&quot;</td>
<td>NONE</td>
</tr>
<tr>
<td>1 or &quot;::errors&quot;</td>
<td>ERRORS</td>
</tr>
<tr>
<td>2 or &quot;::all&quot;</td>
<td>ALL</td>
</tr>
</tbody>
</table>

  The "::xxx" values are symbolic pseudo-constants that may be given instead of the literal numeric values. They must be quoted as strings and are case-sensitive.

- **`audit_log_policy_value`**

  This variable corresponds to the value of the `audit_log_policy` system variable. The value is an unsigned integer. Table 6.24, "`audit_log_policy_value` Values" shows the permitted values and the corresponding `audit_log_policy` values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Corresponding audit_log_policy Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 or &quot;::none&quot;</td>
<td>NONE</td>
</tr>
<tr>
<td>1 or &quot;::logins&quot;</td>
<td>LOGINS</td>
</tr>
<tr>
<td>2 or &quot;::all&quot;</td>
<td>ALL</td>
</tr>
<tr>
<td>3 or &quot;::queries&quot;</td>
<td>QUERIES</td>
</tr>
</tbody>
</table>

  The "::xxx" values are symbolic pseudo-constants that may be given instead of the literal numeric values. They must be quoted as strings and are case-sensitive.

- **`audit_log_statement_policy_value`**

  This variable corresponds to the value of the `audit_log_statement_policy` system variable. The value is an unsigned integer. Table 6.25, "`audit_log_statement_policy_value` Values" shows the permitted values and the corresponding `audit_log_statement_policy` values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Corresponding audit_log_statement_policy Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 or &quot;::none&quot;</td>
<td>NONE</td>
</tr>
<tr>
<td>1 or &quot;::errors&quot;</td>
<td>ERRORS</td>
</tr>
<tr>
<td>Value</td>
<td>Corresponding audit_log_statement_policy Value</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>2 or &quot;::all&quot;</td>
<td>ALL</td>
</tr>
</tbody>
</table>

The "::xxx" values are symbolic pseudo-constants that may be given instead of the literal numeric values. They must be quoted as strings and are case-sensitive.

**Referencing Predefined Functions**

To refer to a predefined function in a log condition, use a function item, which takes name and args values to specify the function name and its arguments, respectively:

```json
{
  "filter": {
    "class": {
      "name": "general",
      "event": {
        "name": "status",
        "log": {
          "function": {
            "name": "find_in_include_list",
            "args": [ { "string": [ { "field": "user.str" }, { "string": '@' }, { "field": "host.str" } ] } ]
          }
        }
      }
    }
  }
}
```

The function as specified in the name item should be the function name only, without parentheses or the argument list. Arguments in the args item, if there is one, must be given in the order listed in the function description. Arguments can refer to predefined variables, event fields, or string or numeric constants.

The preceding filter determines whether to log general class status events depending on whether the current user is found in the audit_log_include_accounts system variable. That user is constructed using fields in the event.

The following list describes the permitted predefined functions for function items:

- **audit_log_exclude_accounts_is_null()**
  Checks whether the audit_log_exclude_accounts system variable is NULL. This function can be helpful when defining filters that correspond to the legacy audit log implementation.
  
  Arguments:
  None.

- **audit_log_include_accounts_is_null()**
  Checks whether the audit_log_include_accounts system variable is NULL. This function can be helpful when defining filters that correspond to the legacy audit log implementation.
  
  Arguments:
  None.

- **debug_sleep(millisec)**
  Sleeps for the given number of milliseconds. This function is used during performance measurement. **debug_sleep()** is available for debug builds only.
Writing Audit Log Filter Definitions

Arguments:

- **millisec**: An unsigned integer that specifies the number of milliseconds to sleep.

- **find_in_exclude_list(account)**
  Checks whether an account string exists in the audit log exclude list (the value of the `audit_log_exclude_accounts` system variable).

Arguments:

- **account**: A string that specifies the user account name.

- **find_in_include_list(account)**
  Checks whether an account string exists in the audit log include list (the value of the `audit_log_include_accounts` system variable).

Arguments:

- **account**: A string that specifies the user account name.

- **string_find(text, substr)**
  Checks whether the `substr` value is contained in the `text` value. This search is case-sensitive.

Arguments:

- **text**: The text string to search.

- **substr**: The substring to search for in `text`.

Replacing a User Filter

In some cases, the filter definition can be changed dynamically. To do this, define a `filter` configuration within an existing `filter`. For example:

```json
{
  "filter": {
    "id": "main",
    "class": {
      "name": "table_access",
      "event": {
        "name": ["update", "delete"],
        "log": false,
        "filter": {
          "class": {
            "name": "general",
            "event": {
              "name": "status",
              "filter": { "ref": "main" } }
          },
          "activate": {
            "or": [
              { "field": { "name": "table_name.str", "value": "temp_1" } },
              { "field": { "name": "table_name.str", "value": "temp_2" } }
            ]
          }
        }
      }
    }
  }
}
```

A new filter is activated when the `activate` element within a subfilter evaluates to `true`. Using `activate` in a top-level `filter` is not permitted.
A new filter can be replaced with the original one by using a ref item inside the subfilter to refer to the original filter id.

The filter shown operates like this:

- The main filter waits for table_access events, either update or delete.
- If the update or delete table_access event occurs on the temp_1 or temp_2 table, the filter is replaced with the internal one (without an id, since there is no need to refer to it explicitly).
- If the end of the command is signalled (general/status event), an entry is written to the audit log file and the filter is replaced with the main filter.

The filter is useful to log statements that update or delete anything from the temp_1 or temp_2 tables, such as this one:

```
UPDATE temp_1, temp_3 SET temp_1.a=21, temp_3.a=23;
```

The statement generates multiple table_access events, but the audit log file will contain only general/status entries.

Note

Any id values used in the definition are evaluated with respect only to that definition. They have nothing to do with the value of the audit_log_filter_id system variable.

6.5.8 Legacy Mode Audit Log Filtering

Note

This section describes legacy audit log filtering, which applies if the audit_log plugin is installed but not the accompanying audit tables and UDFs needed for rule-based filtering.

The audit log plugin can filter audited events. This enables you to control whether audited events are written to the audit log file based on the account from which events originate or event status. Status filtering occurs separately for connection events and statement events.

- Event Filtering by Account
- Event Filtering by Status

Event Filtering by Account

To filter audited events based on the originating account, set one of these system variables at server startup or runtime:

- audit_log_include_accounts: The accounts to include in audit logging. If this variable is set, only these accounts are audited.
- audit_log_exclude_accounts: The accounts to exclude from audit logging. If this variable is set, all but these accounts are audited.

The value for either variable can be NULL or a string containing one or more comma-separated account names, each in user_name@host_name format. By default, both variables are NULL, in which case, no account filtering is done and auditing occurs for all accounts.

Modifications to audit_log_include_accounts or audit_log_exclude_accounts affect only connections created subsequent to the modification, not existing connections.

Example: To enable audit logging only for the user1 and user2 local host account accounts, set the audit_log_include_accounts system variable like this:

```sql
SET SESSION audit_log_include_accounts = 'user1, user2';
```
Legacy Mode Audit Log Filtering

SET GLOBAL audit_log_include_accounts = 'user1@localhost,user2@localhost';

Only one of audit_log_include_accounts or audit_log_exclude_accounts can be
non-NULL at a time:
• If you set audit_log_include_accounts, the server sets audit_log_exclude_accounts to
NULL.
• If you attempt to set audit_log_exclude_accounts, an error occurs unless
audit_log_include_accounts is NULL. In this case, you must first clear
audit_log_include_accounts by setting it to NULL.
-- This sets audit_log_exclude_accounts to NULL
SET GLOBAL audit_log_include_accounts = value;
-- This fails because audit_log_include_accounts is not NULL
SET GLOBAL audit_log_exclude_accounts = value;
-- To set audit_log_exclude_accounts, first set
-- audit_log_include_accounts to NULL
SET GLOBAL audit_log_include_accounts = NULL;
SET GLOBAL audit_log_exclude_accounts = value;

If you inspect the value of either variable, be aware that SHOW VARIABLES displays NULL as an empty
string. To avoid this, use SELECT instead:
mysql> SHOW VARIABLES LIKE 'audit_log_include_accounts';
+----------------------------+-------+
| Variable_name
| Value |
+----------------------------+-------+
| audit_log_include_accounts |
|
+----------------------------+-------+
mysql> SELECT @@audit_log_include_accounts;
+------------------------------+
| @@audit_log_include_accounts |
+------------------------------+
| NULL
|
+------------------------------+

If a user name or host name requires quoting because it contains a comma, space, or other special
character, quote it using single quotes. If the variable value itself is quoted with single quotes, double
each inner single quote or escape it with a backslash. The following statements each enable audit
logging for the local root account and are equivalent, even though the quoting styles differ:
SET
SET
SET
SET

GLOBAL
GLOBAL
GLOBAL
GLOBAL

audit_log_include_accounts
audit_log_include_accounts
audit_log_include_accounts
audit_log_include_accounts

=
=
=
=

'root@localhost';
'''root''@''localhost''';
'\'root\'@\'localhost\'';
"'root'@'localhost'";

The last statement will not work if the ANSI_QUOTES SQL mode is enabled because in that mode
double quotes signify identifier quoting, not string quoting.

Event Filtering by Status
To filter audited events based on status, set the following system variables at server startup or runtime.
These variables apply only for legacy audit log filtering. For JSON audit log filtering, different status
variables apply; see Audit Log Options and Variables.
• audit_log_connection_policy: Logging policy for connection events
• audit_log_statement_policy: Logging policy for statement events
Each variable takes a value of ALL (log all associated events; this is the default), ERRORS (log only
failed events), or NONE (do not log events). For example, to log all statement events but only failed
connection events, use these settings:
SET GLOBAL audit_log_statement_policy = ALL;
SET GLOBAL audit_log_connection_policy = ERRORS;

318


Another policy system variable, `audit_log_policy`, is available but does not afford as much control as `audit_log_connection_policy` and `audit_log_statement_policy`. It can be set only at server startup. At runtime, it is a read-only variable. It takes a value of ALL (log all events; this is the default), LOGINS (log connection events), QUERIES (log statement events), or NONE (do not log events). For any of those values, the audit log plugin logs all selected events without distinction as to success or failure. Use of `audit_log_policy` at startup works as follows:

- If you do not set `audit_log_policy` or set it to its default of ALL, any explicit settings for `audit_log_connection_policy` or `audit_log_statement_policy` apply as specified. If not specified, they default to ALL.
- If you set `audit_log_policy` to a non-ALL value, that value takes precedence over and is used to set `audit_log_connection_policy` and `audit_log_statement_policy`, as indicated in the following table. If you also set either of those variables to a value other than their default of ALL, the server writes a message to the error log to indicate that their values are being overridden.

<table>
<thead>
<tr>
<th>Startup <code>audit_log_policy</code> Value</th>
<th>Resulting <code>audit_log_connection_policy</code> Value</th>
<th>Resulting <code>audit_log_statement_policy</code> Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOGINS</td>
<td>ALL</td>
<td>NONE</td>
</tr>
<tr>
<td>QUERIES</td>
<td>NONE</td>
<td>ALL</td>
</tr>
<tr>
<td>NONE</td>
<td>NONE</td>
<td>NONE</td>
</tr>
</tbody>
</table>

### 6.5.9 Audit Log Reference

The following discussion serves as a reference to MySQL Enterprise Audit components:

- **Audit Log Tables**
- **Audit Log Functions**
- **Audit Log Option and Variable Reference**
- **Audit Log Options and Variables**
- **Audit Log Status Variables**

To install the audit log tables and functions, use the instructions provided in Section 6.5.2, “Installing or Uninstalling MySQL Enterprise Audit”. Unless those components are installed, the `audit_log` plugin operates in legacy mode. See Section 6.5.8, “Legacy Mode Audit Log Filtering”.

**Audit Log Tables**

MySQL Enterprise Audit uses tables in the `mysql` system database for persistent storage of filter and user account data. The tables can be accessed only by users with privileges for that database. The tables use the InnoDB storage engine.

If these tables are missing, the `audit_log` plugin operates in legacy mode. See Section 6.5.8, “Legacy Mode Audit Log Filtering”.

The `audit_log_filter` table stores filter definitions. The table has these columns:

- **NAME**
  The filter name.
- **FILTER**
  The filter definition associated with the filter name. Definitions are stored as JSON values.

The `audit_log_user` table stores user account information. The table has these columns:
Audit Log Reference

- **USER**
  The user name part of an account. For an account `user1@localhost`, the USER part is `user1`.

- **HOST**
  The host name part of an account. For an account `user1@localhost`, the HOST part is `localhost`.

- **FILTERNAME**
  The name of the filter assigned to the account. The filter name associates the account with a filter defined in the `audit_log_filter` table.

Audit Log Functions

This section describes, for each audit log user-defined function (UDF), its purpose, calling sequence, and return value. For information about the conditions under which these UDFs can be invoked, see Section 6.5.6, “Audit Log Filtering”.

Each audit log UDF returns a string that indicates whether the operation succeeded. **OK** indicates success. **ERROR: message** indicates failure.

As of MySQL 8.0.19, audit log UDFs convert string arguments to `utf8mb4` and string return values are `utf8mb4` strings. Prior to MySQL 8.0.19, audit log UDFs treat string arguments as binary strings (which means they do not distinguish lettercase), and string return values are binary strings.

These audit log UDFs are available:

- **audit_log_encryption_password_get([keyring_id])**

  This function fetches an audit log encryption password from the MySQL keyring, which must be enabled or an error occurs. Any keyring plugin can be used; for instructions, see Section 6.4, “The MySQL Keyring”.

  With no argument, the function retrieves the current encryption password as a binary string. An argument may be given to specify which audit log encryption password to retrieve. The argument must be the keyring ID of the current password or an archived password.

  For additional information about audit log encryption, see Audit Log File Encryption.

  Arguments:

  - **keyring_id**: As of MySQL 8.0.17, this optional argument indicates the keyring ID of the password to retrieve. The maximum permitted length is 766 bytes. If omitted, the function retrieves the current password.

  Prior to MySQL 8.0.17, no argument is permitted. The function always retrieves the current password.

  Return value:

  The password string for success (up to 766 bytes), or **NULL** and an error for failure.

  Example:

  Retrieve the current password:

  ```sql
  mysql> SELECT audit_log_encryption_password_get();
  +--------------------------------------+
  | audit_log_encryption_password_get() |
  +--------------------------------------+
  320
  ```
To retrieve a password by ID, you can determine which audit log keyring IDs exist by querying the Performance Schema `keyring_keys` table:

```
mysql> SELECT * FROM performance_schema.keyring_keys
WHERE KEY_ID LIKE 'audit_log%'
ORDER BY KEY_ID;

+-----------------------------+-----------+----------------+
| KEY_ID                      | KEY_OWNER | BACKEND_KEY_ID |
+-----------------------------+-----------+----------------+
| audit_log-20190415T152248-1 |           |                |
| audit_log-20190415T153507-1 |           |                |
| audit_log-20190416T125122-1 |           |                |
| audit_log-20190416T141608-1 |           |                |
+-----------------------------+-----------+----------------+
mysql>
```

```
mysql> SELECT audit_log_encryption_password_get('audit_log-20190416T125122-1');
+------------------------------------------------------------------+
| audit_log_encryption_password_get('audit_log-20190416T125122-1') |
+------------------------------------------------------------------+
| segreto                                                          |
+------------------------------------------------------------------+
```

- `audit_log_encryption_password_set(password)`

  Sets the current audit log encryption password to the argument and stores the password in the MySQL keyring. As of MySQL 8.0.19, the password is stored as a `utf8mb4` string. Prior to MySQL 8.0.19, the password is stored in binary form.

  If encryption is enabled, this function performs a log file rotation operation that renames the current log file, and begins a new log file encrypted with the password. The keyring must be enabled or an error occurs. Any keyring plugin can be used; for instructions, see Section 6.4, “The MySQL Keyring”.

  For additional information about audit log encryption, see Audit Log File Encryption.

  Arguments:

  `password`: The password string. The maximum permitted length is 766 bytes.

  Return value:

  1 for success, 0 for failure.

  Example:

  ```
  mysql> SELECT audit_log_encryption_password_set(password);
  +---------------------------------------------+
  | audit_log_encryption_password_set(password) |
  +---------------------------------------------+
  | 1                                           |
  +---------------------------------------------+
  ```

- `audit_log_filter_flush()`

  Calling any of the other filtering UDFs affects operational audit log filtering immediately and updates the audit log tables. If instead you modify the contents of those tables directly using statements such as `INSERT`, `UPDATE`, and `DELETE`, the changes do not affect filtering immediately. To flush your changes and make them operational, call `audit_log_filter_flush()`.

  **Warning**

  `audit_log_filter_flush()` should be used only after modifying the audit tables directly, to force reloading all filters. Otherwise, this function should be avoided. It is, in effect, a simplified version of unloading and reloading.
reloading the `audit_log` plugin with `UNINSTALL PLUGIN` plus `INSTALL PLUGIN`.

`audit_log_filter_flush()` affects all current sessions and detaches them from their previous filters. Current sessions are no longer logged unless they disconnect and reconnect, or execute a change-user operation.

If this function fails, an error message is returned and the audit log is disabled until the next successful call to `audit_log_filter_flush()`.

Arguments:
None.

Return value:
A string that indicates whether the operation succeeded. `OK` indicates success. `ERROR: message` indicates failure.

Example:
```
mysql> SELECT audit_log_filter_flush();
+--------------------------+
| audit_log_filter_flush() |
| OK                       |
+--------------------------+
```

• `audit_log_filter_remove_filter(filter_name)`

Given a filter name, removes the filter from the current set of filters. It is not an error for the filter not to exist.

If a removed filter is assigned to any user accounts, those users stop being filtered (they are removed from the `audit_log_user` table). Termination of filtering includes any current sessions for those users: They are detached from the filter and no longer logged.

Arguments:

• `filter_name`: A string that specifies the filter name.

Return value:
A string that indicates whether the operation succeeded. `OK` indicates success. `ERROR: message` indicates failure.

Example:
```
mysql> SELECT audit_log_filter_remove_filter('SomeFilter');
+----------------------------------------------+
| audit_log_filter_remove_filter('SomeFilter') |
| OK                                           |
+----------------------------------------------+
```

• `audit_log_filter_remove_user(user_name)`

Given a user account name, cause the user to be no longer assigned to a filter. It is not an error if the user has no filter assigned. Filtering of current sessions for the user remains unaffected. New connections for the user are filtered using the default account filter if there is one, and are not logged otherwise.

If the name is `%`, the function removes the default account filter that is used for any user account that has no explicitly assigned filter.
Arguments:

- `user_name`: The user account name as a string in `user_name@host_name` format, or `%` to represent the default account.

Return value:

A string that indicates whether the operation succeeded. `OK` indicates success. `ERROR: message` indicates failure.

Example:

```
mysql> SELECT audit_log_filter_remove_user('user1@localhost');
+-------------------------------------------------+
| audit_log_filter_remove_user('user1@localhost') |
| OK                                              |
+-------------------------------------------------+
```

- `audit_log_filter_set_filter(filter_name, definition)`

Given a filter name and definition, adds the filter to the current set of filters. If the filter already exists and is used by any current sessions, those sessions are detached from the filter and are no longer logged. This occurs because the new filter definition has a new filter ID that differs from its previous ID.

Arguments:

- `filter_name`: A string that specifies the filter name.
- `definition`: A JSON value that specifies the filter definition.

Return value:

A string that indicates whether the operation succeeded. `OK` indicates success. `ERROR: message` indicates failure.

Example:

```
mysql> SET @f = '{ "filter": { "log": false } }';
mysql> SELECT audit_log_filter_set_filter('SomeFilter', @f);  
+-----------------------------------------------+
| audit_log_filter_set_filter('SomeFilter', @f) |
| OK                                            |
+-----------------------------------------------+
```
• **audit_log_filter_set_user**(user_name, filter_name)

Given a user account name and a filter name, assigns the filter to the user. A user can be assigned only one filter, so if the user was already assigned a filter, the assignment is replaced. Filtering of current sessions for the user remains unaffected. New connections are filtered using the new filter.

As a special case, the name `%` represents the default account. The filter is used for connections from any user account that has no explicitly assigned filter.

**Arguments:**

- **user_name**: The user account name as a string in `user_name@host_name` format, or `%` to represent the default account.
- **filter_name**: A string that specifies the filter name.

**Return value:**

A string that indicates whether the operation succeeded. OK indicates success. ERROR: message indicates failure.

**Example:**

```
mysql> SELECT audit_log_filter_set_user('user1@localhost', 'SomeFilter');
```

```
+------------------------------------------------------------+
| audit_log_filter_set_user('user1@localhost', 'SomeFilter') |
+------------------------------------------------------------+
| OK                                                         |
+------------------------------------------------------------+
```

• **audit_log_read**(arg)

Reads events from the audit log and returns a binary JSON string containing an array of audit events. If the audit log format is not JSON, an error occurs.

Each event in the return value is a JSON hash, except that the last array element may be a JSON null value to indicate no following events are available to read.

For the first call to `audit_log_read()` within a session, pass a bookmark indicating where to begin reading. If the final value of the returned array is not a JSON null value, there are more events following those just read and `audit_log_read()` can be called without or with a bookmark argument. Without an argument, reading continues with the next unread event. With a bookmark argument, reading continues from the bookmark.

If the final value of the returned array is a JSON null value, there are no more events left to be read and the next call to `audit_log_read()` must include a bookmark argument.

To obtain a bookmark for the most recently written event, call `audit_log_read_bookmark()`.

For additional information about audit log-reading functions, see Audit Log File Reading.

**Arguments:**

- **arg**: An optional bookmark, represented as a string containing a JSON hash that indicates where and how much to read. The following items are significant in the arg value (other items are ignored):
  - **timestamp, id**: The location within the audit log of the first event to read. Both items must be present to completely specify a position.
  - **max_array_length**: The maximum number of events to read from the log. If omitted, the default is to read to the end of the log or until the read buffer is full, whichever comes first.

**Return value:**
A binary JSON string containing an array of audit events for success, or NULL and an error for failure.

Example:
```
mysql> SELECT audit_log_read(audit_log_read_bookmark());
+-----------------------------------------------------------------------+
| audit_log_read(audit_log_read_bookmark())                             |
+-----------------------------------------------------------------------+
| [ { "timestamp": "2019-10-03 22:41:24", "id": 0, "class": "connection" } ] |
+-----------------------------------------------------------------------+
```

• `audit_log_read_bookmark()`

Returns a binary JSON string representing a bookmark for the most recently written audit log event. If the audit log format is not JSON, an error occurs.

The bookmark is a JSON hash with `timestamp` and `id` items indicating the event position within the audit log. It is suitable for passing to `audit_log_read()` to indicate to that function where to begin reading.

For additional information about audit log-reading functions, see Audit Log File Reading.

Arguments: None.

Return value: A binary JSON string containing a bookmark for success, or NULL and an error for failure.

Example:
```
mysql> SELECT audit_log_read_bookmark();
+-------------------------------------------------+
| audit_log_read_bookmark()                       |
+-------------------------------------------------+
| { "timestamp": "2019-10-03 21:03:44", "id": 0 } |
+-------------------------------------------------+
```

Audit Log Option and Variable Reference

Table 6.26 Audit Log Option and Variable Reference

<table>
<thead>
<tr>
<th>Name</th>
<th>Cmd-Line</th>
<th>Option File</th>
<th>System Var</th>
<th>Status Var</th>
<th>Var Scope</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit-log</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_log_buffer_size</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_log_compression</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_log_connection_policy</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit_log_current_session</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td>Both</td>
<td>No</td>
</tr>
<tr>
<td>Audit_log_current_size</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td>Global</td>
<td>No</td>
</tr>
<tr>
<td>audit_log_encryption</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td>Global</td>
<td>No</td>
</tr>
<tr>
<td>Audit_log_event_max_drop_size</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td>Global</td>
<td>No</td>
</tr>
<tr>
<td>Audit_log_events</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td>Global</td>
<td>No</td>
</tr>
<tr>
<td>Audit_log_events_filtered</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td>Global</td>
<td>No</td>
</tr>
<tr>
<td>Audit_log_events_lost</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td>Global</td>
<td>No</td>
</tr>
<tr>
<td>Audit_log_events_written</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td>Global</td>
<td>No</td>
</tr>
<tr>
<td>audit_log_exclude_accounts</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

325
Audit Log Reference

### Audit Log Options and Variables

This section describes the command options and system variables that configure operation of MySQL Enterprise Audit. If values specified at startup time are incorrect, the audit_log plugin may fail to initialize properly and the server does not load it. In this case, the server may also produce error messages for other audit log settings because it will not recognize them.

To configure activation of the audit log plugin, use this option:

- `--audit-log[=value]`

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--audit-log[=value]</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>Type</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Default Value</td>
<td>ON</td>
</tr>
<tr>
<td>Valid Values</td>
<td>ON, OFF, FORCE, FORCE_PLUS_PERMANENT</td>
</tr>
</tbody>
</table>

This option controls how the server loads the audit_log plugin at startup. It is available only if the plugin has been previously registered with `INSTALL PLUGIN` or is loaded with `--plugin-load` or `--plugin-load-add`. See Section 6.5.2, “Installing or Uninstalling MySQL Enterprise Audit”.

The option value should be one of those available for plugin-loading options, as described in `Installing and Uninstalling Plugins`. For example, `--audit-log=FORCE_PLUS_PERMANENT` tells the server to load the plugin and prevent it from being removed while the server is running.

If the audit log plugin is enabled, it exposes several system variables that permit control over logging:

```sql
mysql> SHOW VARIABLES LIKE 'audit_log%';
+-----------------------------+--------------+
| Variable_name               | Value        |
+-----------------------------+--------------+
| audit_log_buffer_size       | 1048576      |
+-----------------------------+--------------+
```

---

**Audit Log Options and Variables**

**Name** | **Cmd-Line** | **Option File** | **System Var** | **Status Var** | **Var Scope** | **Dynamic**
---|---|---|---|---|---|---
audit_log_file | Yes | Yes | Yes | Global | No |
audit_log_filter_id | Yes | Yes | Yes | Global | No |
audit_log_flush | Yes | Yes | Yes | Global | Yes |
audit_log_format | Yes | Yes | Yes | Global | No |
audit_log_include_accounts | Yes | Yes | Yes | Global | Yes |
audit_log_password_history_days | Yes | Yes | Yes | Global | Yes |
audit_log_policy | Yes | Yes | Yes | Global | No |
audit_log_read_buffer_size | Yes | Yes | Yes | Varies | Varies |
audit_log_rotate_on_size | Yes | Yes | Yes | Global | Yes |
audit_log_statement_policy | Yes | Yes | Yes | Global | Yes |
audit_log_strategy | Yes | Yes | Yes | Global | No |
Audit_log_total_size | Yes | Yes | Global | No |
Audit_log_write_waits | Yes | Yes | Global | No |
### Audit Log Reference

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit_log_connection_policy</td>
<td>ALL</td>
</tr>
<tr>
<td>audit_log_current_session</td>
<td>OFF</td>
</tr>
<tr>
<td>audit_logExclude_accounts</td>
<td>OFF</td>
</tr>
<tr>
<td>audit_log_file</td>
<td>audit.log</td>
</tr>
<tr>
<td>audit_log_filter_id</td>
<td>0</td>
</tr>
<tr>
<td>audit_log_flush</td>
<td>OFF</td>
</tr>
<tr>
<td>audit_log_format</td>
<td>NEW</td>
</tr>
<tr>
<td>audit_log_include_accounts</td>
<td>OFF</td>
</tr>
<tr>
<td>audit_log_policy</td>
<td>ALL</td>
</tr>
<tr>
<td>audit_log_rotate_on_size</td>
<td>0</td>
</tr>
<tr>
<td>audit_log_statement_policy</td>
<td>ALL</td>
</tr>
<tr>
<td>audit_log_strategy</td>
<td>ASYNCHRONOUS</td>
</tr>
</tbody>
</table>

You can set any of these variables at server startup, and some of them at runtime. Those that are available only for legacy mode audit log filtering are so noted.

- **audit_log_buffer_size**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td>--audit-log-buffer-size=#</td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td>audit_log_buffer_size</td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>No</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>1048576</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>4096</td>
</tr>
<tr>
<td>Maximum Value (64-bit platforms)</td>
<td>18446744073709547520</td>
</tr>
<tr>
<td>Maximum Value (32-bit platforms)</td>
<td>4294967295</td>
</tr>
</tbody>
</table>

When the audit log plugin writes events to the log asynchronously, it uses a buffer to store event contents prior to writing them. This variable controls the size of that buffer, in bytes. The server adjusts the value to a multiple of 4096. The plugin uses a single buffer, which it allocates when it initializes and removes when it terminates. The plugin allocates this buffer only if logging is asynchronous.

- **audit_log_compression**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td>--audit-log-compression=value</td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td>audit_log_compression</td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>No</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Default Value</td>
<td>NONE</td>
</tr>
<tr>
<td>Valid Values</td>
<td>NONE, GZIP</td>
</tr>
</tbody>
</table>
The type of compression for the audit log file. Permitted values are **NONE** (no compression; the default) and **GZIP** (GNU Zip compression). For more information, see *Audit Log File Compression*.

- **audit_log_connection_policy**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--audit-log-connection-policy=value</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>audit_log_connection_policy</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Default Value</td>
<td><strong>ALL</strong></td>
</tr>
<tr>
<td>Valid Values</td>
<td><strong>ALL</strong></td>
</tr>
<tr>
<td></td>
<td><strong>ERRORS</strong></td>
</tr>
<tr>
<td></td>
<td><strong>NONE</strong></td>
</tr>
</tbody>
</table>

**Note**

This variable applies only to legacy mode audit log filtering (see *Section 6.5.8, “Legacy Mode Audit Log Filtering”*).

The policy controlling how the audit log plugin writes connection events to its log file. The following table shows the permitted values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ALL</strong></td>
<td>Log all connection events</td>
</tr>
<tr>
<td><strong>ERRORS</strong></td>
<td>Log only failed connection events</td>
</tr>
<tr>
<td><strong>NONE</strong></td>
<td>Do not log connection events</td>
</tr>
</tbody>
</table>

**Note**

At server startup, any explicit value given for `audit_log_connection_policy` may be overridden if `audit_log_policy` is also specified, as described in *Section 6.5.5, “Audit Log Logging Configuration”*.  

- **audit_log_current_session**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>audit_log_current_session</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global, Session</td>
</tr>
<tr>
<td>Dynamic</td>
<td>No</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
</tbody>
</table>
### Audit Log Reference

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>depends on filtering policy</td>
</tr>
</tbody>
</table>

Whether audit logging is enabled for the current session. The session value of this variable is read only. It is set when the session begins based on the values of the `audit_log_include_accounts` and `audit_log_exclude_accounts` system variables. The audit log plugin uses the session value to determine whether to audit events for the session. (There is a global value, but the plugin does not use it.)

- **audit_log_encryption**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--audit-log-encryption=value</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>audit_log_encryption</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>No</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Default Value</td>
<td><code>NONE</code></td>
</tr>
<tr>
<td>Valid Values</td>
<td><code>NONE</code> <code>AES</code></td>
</tr>
</tbody>
</table>

The type of encryption for the audit log file. Permitted values are `NONE` (no encryption; the default) and `AES` (AES-256-CBC cipher encryption). For more information, see [Audit Log File Encryption](#).

- **audit_log_exclude_accounts**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--audit-log-exclude-accounts=value</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>audit_log_exclude_accounts</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td><code>NULL</code></td>
</tr>
</tbody>
</table>

**Note**

This variable applies only to legacy mode audit log filtering (see Section 6.5.8, "Legacy Mode Audit Log Filtering").

The accounts for which events should not be logged. The value should be `NULL` or a string containing a list of one or more comma-separated account names. For more information, see Section 6.5.6, "Audit Log Filtering".

Modifications to `audit_log_exclude_accounts` affect only connections created subsequent to the modification, not existing connections.
• **audit_log_file**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--audit-log-file=file_name</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>audit_log_file</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>No</td>
</tr>
<tr>
<td><strong>SET_VAR</strong> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>File name</td>
</tr>
<tr>
<td>Default Value</td>
<td><code>audit.log</code></td>
</tr>
</tbody>
</table>

The base name and suffix of the file to which the audit log plugin writes events. The default value is `audit.log`, regardless of logging format. To have the name suffix correspond to the format, set the name explicitly, choosing a different suffix (for example, `audit.xml` for XML format, `audit.json` for JSON format).

If the value of `audit_log_file` is a relative path name, the plugin interprets it relative to the data directory. If the value is a full path name, the plugin uses the value as is. A full path name may be useful if it is desirable to locate audit files on a separate file system or directory. For security reasons, write the audit log file to a directory accessible only to the MySQL server and to users with a legitimate reason to view the log.

For details about how the audit log plugin interprets the `audit_log_file` value and the rules for file renaming that occurs at plugin initialization and termination, see Audit Log File Name.

The audit log plugin uses the directory containing the audit log file (determined from the `audit_log_file` value) as the location to search for readable audit log files. From these log files and the current file, the plugin constructs a list of the ones that are subject to use with the audit log bookmarking and reading functions. See Audit Log File Reading.

• **audit_log_filter_id**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>audit_log_filter_id</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global, Session</td>
</tr>
<tr>
<td>Dynamic</td>
<td>No</td>
</tr>
<tr>
<td><strong>SET_VAR</strong> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
</tbody>
</table>

The session value of this variable indicates the internally maintained ID of the audit filter for the current session. A value of 0 means that the session has no filter assigned.

• **audit_log_flush**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>audit_log_flush</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
</tbody>
</table>
When this variable is set to enabled (1 or ON), the audit log plugin closes and reopens its log file to flush it. (The value remains OFF so that you need not disable it explicitly before enabling it again to perform another flush.) Enabling this variable has no effect unless audit_log_rotate_on_size is 0. For more information, see Section 6.5.5, “Audit Log Logging Configuration”.

- **audit_log_format**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td>--audit-log-format=value</td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td>audit_log_format</td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>No</td>
</tr>
<tr>
<td><strong>SET_VAR</strong> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Default Value</td>
<td>NEW</td>
</tr>
<tr>
<td>Valid Values</td>
<td>OLD, NEW, JSON</td>
</tr>
</tbody>
</table>

The audit log file format. Permitted values are OLD (old-style XML), NEW (new-style XML; the default), and JSON. For details about each format, see Section 6.5.4, “Audit Log File Formats”.

Note

For information about issues to consider when changing the log format, see Audit Log File Format.
This variable applies only to legacy mode audit log filtering (see Section 6.5.8, “Legacy Mode Audit Log Filtering”).

The accounts for which events should be logged. The value should be NULL or a string containing a list of one or more comma-separated account names. For more information, see Section 6.5.6, “Audit Log Filtering”.

Modifications to audit_log_include_accounts affect only connections created subsequent to the modification, not existing connections.

• audit_log_password_history_keep_days

When a new encryption password is created, the audit log plugin archives the previous password, if one exists, for later use. The audit_log_password_history_keep_days variable controls automatic removal of archived passwords. Its value indicates the number of days after which archived audit log encryption passwords are removed. The default is 0, which disables password removal; the password retention period is forever.

New audit log encryption passwords are created under these circumstances:

• During plugin initialization, if the plugin finds that log file encryption is enabled, it checks whether the keyring contains an audit log encryption password. If not, the plugin automatically generates a random initial encryption password.

• When the audit_log_encryption_password_set() function is called to set a specific password.

In each case, the plugin stores the password in the key ring and uses it to encrypt new log files.

Cleanup of old audit log encryption passwords occurs under these circumstances:

• During plugin initialization.

• When the audit_log_encryption_password_set() function is called.

• When the runtime value of audit_log_password_history_keep_days is changed from its current value to a value greater than 0. Runtime value changes may occur for SET statements
that use the `GLOBAL` or `PERSIST` keyword, but not the `PERSISTONLY` keyword. `PERSISTONLY` writes the variable setting to `mysqld-auto.cnf`, but has no effect on the runtime value.

When password cleanup occurs, the current value of `audit_log_password_history_keep_days` determines which passwords to remove:

- If the value is $N > 0$, the plugin removes passwords more than $N$ days old.
- If the value is 0, the plugin removes no passwords.

**Note**

Take care not to expire old passwords that are still needed to read archived encrypted log files.

If you normally leave password expiration disabled (`audit_log_password_history_keep_days=0`), it is possible to perform an on-demand cleanup operation by temporarily assigning the variable a value greater than zero. For example, to expire passwords older than 365 days, do this:

```
SET GLOBAL audit_log_password_history_keep_days = 365;
SET GLOBAL audit_log_password_history_keep_days = 0;
```

Setting the runtime value of `audit_log_password_history_keep_days` requires the `AUDIT_ADMIN` privilege, in addition to the `SYSTEM_VARIABLES_ADMIN` or `SUPER` privilege normally required to set a global system variable runtime value.

- `audit_log_policy`

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--audit-log-policy=value</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>audit_log_policy</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>No</td>
</tr>
<tr>
<td><code>SET_VAR</code> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Default Value</td>
<td><code>ALL</code></td>
</tr>
<tr>
<td>Valid Values</td>
<td><code>ALL</code></td>
</tr>
<tr>
<td></td>
<td><code>LOGINS</code></td>
</tr>
<tr>
<td></td>
<td><code>QUERIES</code></td>
</tr>
<tr>
<td></td>
<td><code>NONE</code></td>
</tr>
</tbody>
</table>

**Note**

This variable applies only to legacy mode audit log filtering (see Section 6.5.8, “Legacy Mode Audit Log Filtering”).

The policy controlling how the audit log plugin writes events to its log file. The following table shows the permitted values.
### Audit Log Reference

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOGINS</td>
<td>Log only login events</td>
</tr>
<tr>
<td>QUERIES</td>
<td>Log only query events</td>
</tr>
<tr>
<td>NONE</td>
<td>Log nothing (disable the audit stream)</td>
</tr>
</tbody>
</table>

**audit_log_policy** can be set only at server startup. At runtime, it is a read-only variable. Two other system variables, **audit_log_connection_policy** and **audit_log_statement_policy**, provide finer control over logging policy and can be set either at startup or at runtime. If you use **audit_log_policy** at startup instead of the other two variables, the server uses its value to set those variables. For more information about the policy variables and their interaction, see Section 6.5.5, “Audit Log Logging Configuration”.

- **audit_log_read_buffer_size**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--audit-log-read-buffer-size=#</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable (≥ 8.0.11)</td>
<td><code>audit_log_read_buffer_size</code></td>
</tr>
<tr>
<td>Scope (≥ 8.0.12)</td>
<td>Global, Session</td>
</tr>
<tr>
<td>Scope (8.0.11)</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic (≥ 8.0.12)</td>
<td>Yes</td>
</tr>
<tr>
<td>Dynamic (8.0.11)</td>
<td>No</td>
</tr>
<tr>
<td>SET_VAR Hint Applies (≥ 8.0.11)</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value (≥ 8.0.12)</td>
<td>32768</td>
</tr>
<tr>
<td>Default Value (8.0.11)</td>
<td>1048576</td>
</tr>
<tr>
<td>Minimum Value (≥ 8.0.12)</td>
<td>32768</td>
</tr>
<tr>
<td>Minimum Value (8.0.11)</td>
<td>1024</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>4194304</td>
</tr>
</tbody>
</table>

The buffer size for reading from the audit log file, in bytes. The `audit_log_read()` function reads no more than this many bytes. Log file reading is supported only for JSON log format. For more information, see Audit Log File Reading.

As of MySQL 8.0.12, this variable has a default of 32KB and can be set at runtime. Each client should set its session value of `audit_log_read_buffer_size` appropriately for its use of `audit_log_read()`. Prior to MySQL 8.0.12, `audit_log_read_buffer_size` has a default of 1MB, affects all clients, and can be changed only at server startup.

- **audit_log_rotate_on_size**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--audit-log-rotate-on-size=#</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>audit_log_rotate_on_size</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
</tbody>
</table>

The buffer size for rotating the audit log file, in bytes. The log file will be rotated when it reaches this size. Log file rotation is supported only for JSON log format. For more information, see Audit Log File Reading.
### Audit Log Reference

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>0</td>
</tr>
</tbody>
</table>

If the `audit_log_rotate_on_size` value is 0, the audit log plugin does not perform automatic log file rotation. Instead, use `audit_log_flush` to close and reopen the log on demand. In this case, manually rename the file externally to the server before flushing it.

If the `audit_log_rotate_on_size` value is greater than 0, automatic size-based log file rotation occurs. Whenever a write to the log file causes its size to exceed the `audit_log_rotate_on_size` value, the audit log plugin closes the current log file, renames it, and opens a new log file.

For more information about audit log file rotation, see [Audit Log File Space Management and Name Rotation](#).

If you set this variable to a value that is not a multiple of 4096, it is truncated to the nearest multiple. (Thus, setting it to a value less than 4096 has the effect of setting it to 0 and no rotation occurs, except manually.)

- **audit_log_statement_policy**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--audit-log-statement-policy=value</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>audit_log_statement_policy</code></td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>SET_VAR</strong> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Default Value</td>
<td>ALL</td>
</tr>
<tr>
<td>Valid Values</td>
<td>ALL, ERRORS, NONE</td>
</tr>
</tbody>
</table>

**Note**

This variable applies only to legacy mode audit log filtering (see [Section 6.5.8, “Legacy Mode Audit Log Filtering”](#)).

The policy controlling how the audit log plugin writes statement events to its log file. The following table shows the permitted values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ALL</strong></td>
<td>Log all statement events</td>
</tr>
<tr>
<td>ERRORS</td>
<td>Log only failed statement events</td>
</tr>
<tr>
<td>NONE</td>
<td>Do not log statement events</td>
</tr>
</tbody>
</table>

**Note**

At server startup, any explicit value given for `audit_log_statement_policy` may be overridden if...
Audit Log Reference

- **audit_log_policy** is also specified, as described in Section 6.5.5, “Audit Log Logging Configuration”.

  - **audit_log_strategy**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td><code>--audit-log-strategy=value</code></td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>audit_log_strategy</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>No</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Default Value</td>
<td><code>ASYNCHRONOUS</code></td>
</tr>
<tr>
<td>Valid Values</td>
<td><code>ASYNCHRONOUS</code></td>
</tr>
<tr>
<td></td>
<td><code>PERFORMANCE</code></td>
</tr>
<tr>
<td></td>
<td><code>SEMISYNCHRONOUS</code></td>
</tr>
<tr>
<td></td>
<td><code>SYNCHRONOUS</code></td>
</tr>
</tbody>
</table>

The logging method used by the audit log plugin. These strategy values are permitted:

- **ASYNCHRONOUS**: Log asynchronously. Wait for space in the output buffer.
- **PERFORMANCE**: Log asynchronously. Drop requests for which there is insufficient space in the output buffer.
- **SEMISYNCHRONOUS**: Log synchronously. Permit caching by the operating system.
- **SYNCHRONOUS**: Log synchronously. Call `sync()` after each request.

**Audit Log Status Variables**

If the audit log plugin is enabled, it exposes several status variables that provide operational information. These variables are available for legacy mode audit filtering and JSON mode audit filtering.

- **Audit_log_current_size**

  The size of the current audit log file. The value increases when an event is written to the log and is reset to 0 when the log is rotated.

- **Audit_log_event_max_drop_size**

  The size of the largest dropped event in performance logging mode. For a description of logging modes, see Section 6.5.5, “Audit Log Logging Configuration”.

- **Audit_log_events**

  The number of events handled by the audit log plugin, whether or not they were written to the log based on filtering policy (see Section 6.5.5, “Audit Log Logging Configuration”).

- **Audit_log_events_filtered**

  The number of events handled by the audit log plugin that were filtered (not written to the log) based on filtering policy (see Section 6.5.5, “Audit Log Logging Configuration”).
6.5.10 Audit Log Restrictions

MySQL Enterprise Audit is subject to these general restrictions:

- Only SQL statements are logged. Changes made by no-SQL APIs, such as memcached, Node.js, and the NDB API, are not logged.
- Only top-level statements are logged, not statements within stored programs such as triggers or stored procedures.
- Contents of files referenced by statements such as LOAD DATA are not logged.

**NDB Cluster.** It is possible to use MySQL Enterprise Audit with MySQL NDB Cluster, subject to the following conditions:

- All changes to be logged must be done using the SQL interface. Changes using no-SQL interfaces, such as those provided by the NDB API, memcached, or ClusterJ, are not logged.
- The plugin must be installed on each MySQL server that is used to execute SQL on the cluster.
- Audit plugin data must be aggregated amongst all MySQL servers used with the cluster. This aggregation is the responsibility of the application or user.

6.6 The Audit Message Component

As of MySQL 8.0.14, the audit_api_message_emit component enables applications to add their own message events to the audit log, using the audit_api_message_emit_udf() user-defined function.

The audit_api_message_emit component cooperates with all plugins of audit type. For concreteness, examples use the audit_log plugin described in Section 6.5, “MySQL Enterprise Audit”.

- Installing or Uninstalling the Audit Message Component
- Audit Message Function

Installing or Uninstalling the Audit Message Component

To be usable by the server, the component library file must be located in the MySQL plugin directory (the directory named by the plugin_dir system variable). If necessary, configure the plugin directory location by setting the value of plugin_dir at server startup.
To install the `audit_api_message_emit` component, use this statement:

```
INSTALL COMPONENT "file://component_audit_api_message_emit";
```

Component installation is a one-time operation that need not be done per server startup. `INSTALL COMPONENT` loads the component, and also registers it in the `mysql.component` system table to cause it to be loaded during subsequent server startups.

To uninstall the `audit_api_message_emit` component, use this statement:

```
UNINSTALL COMPONENT "file://component_audit_api_message_emit";
```

`UNINSTALL COMPONENT` unloads the component, and deregisters it from the `mysql.component` system table to cause it not to be loaded during subsequent server startups.

Installing or uninstalling the `audit_api_message_emit` component installs or uninstalls the `audit_api_message_emit_udf()` function that the component implements. It is not necessary to use `CREATE FUNCTION` or `DROP FUNCTION` to do so.

---

**Audit Message Function**

This section describes the `audit_api_message_emit_udf()` user-defined function (UDF) implemented by the `audit_api_message_emit` component.

Before using the audit message function, install the audit message component according to the instructions provided at Installing or Uninstalling the Audit Message Component.

- `audit_api_message_emit_udf(component, producer, message[, key, value] ...)`
  
  Adds a message event to the audit log. Message events include component, producer, and message strings of the caller's choosing, and optionally a set of key-value pairs.

  An event posted by this UDF is sent to all enabled plugins of audit type, each of which handles the event according to its own rules. If no plugin of audit type is enabled, posting the event has no effect.

  Arguments:

  - `component`: A string that specifies a component name.
  - `producer`: A string that specifies a producer name.
  - `message`: A string that specifies the event message.
  - `key, value`: Events may include 0 or more key-value pairs that specify an arbitrary application-provided data map. Each `key` argument is a string that specifies a name for its immediately following `value` argument. Each `value` argument specifies a value for its immediately following `key` argument. Each `value` can be a string or numeric value, or `NULL`.

  Return value:

  The string `OK` to indicate success. An error occurs if the function fails.

  Example:

  ```
  mysql> SELECT audit_api_message_emit_udf('component_text',
  'producer_text',
  'message_text',
  'key1', 'value1',
  'key2', 123,
  'key3', NULL) AS 'Message';
  +---------+
  | Message |
  +---------+
  | OK      |
  +---------+
  ```

---

338
Audit Message Function

Additional information:

Each audit plugin that receives an event posted by `audit_api_message_emit_udf()` logs the event in plugin-specific format. For example, the `audit_log` plugin (see Section 6.5, “MySQL Enterprise Audit”) logs message values as follows, depending on the log format configured by the `audit_log_format` system variable:

- **JSON format** (`audit_log_format=JSON`):

  ```json
  {
  ...
  "class": "message",
  "event": "user",
  ...
  "message_data": {
  "component": "component_text",
  "producer": "producer_text",
  "message": "message_text",
  "map": {
  "key1": "value1",
  "key2": 123,
  "key3": null
  }
  }
  }
  ```

- **New-style XML format** (`audit_log_format=NEW`):

  ```xml
  <AUDIT_RECORD>
  ...
  <NAME>Message</NAME>
  ...
  <COMMAND_CLASS>user</COMMAND_CLASS>
  <COMPONENT>component_text</COMPONENT>
  <PRODUCER>producer_text</PRODUCER>
  <MESSAGE>message_text</MESSAGE>
  <MAP>
  <ELEMENT>
  <KEY>key1</KEY>
  <VALUE>value1</VALUE>
  </ELEMENT>
  <ELEMENT>
  <KEY>key2</KEY>
  <VALUE>123</VALUE>
  </ELEMENT>
  <ELEMENT>
  <KEY>key3</KEY>
  <VALUE/>
  </ELEMENT>
  </MAP>
  </AUDIT_RECORD>
  ```

- **Old-style XML format** (`audit_log_format=OLD`):

  ```xml
  <AUDIT_RECORD>
  ...
  NAME="Message"
  ...
  COMMAND_CLASS="user"
  COMPONENT="component_text"
  PRODUCER="producer_text"
  ```
Note

Message events logged in old-style XML format do not include the key-value map due to representational constraints imposed by this format.

Messages posted by `audit_api_message_emit_udf()` have an event class of `MYSQL_AUDIT_MESSAGE_CLASS` and a subclass of `MYSQL_AUDIT_MESSAGE_USER`.

(Interally generated audit messages have the same class and a subclass of `MYSQL_AUDIT_MESSAGE_INTERNAL`; this subclass currently is unused.) To refer to such events in `audit_log` filtering rules, use a `class` element with a `name` value of `message`. For example:

```json
{
  "filter": {
    "class": {
      "name": "message"
    }
  }
}
```

Should it be necessary to distinguish user-generated and internally generated message events, test the `subclass` value against `user` or `internal`.

Filtering based on the contents of the key-value map is not supported.

For information about writing filtering rules, see Section 6.5.6, “Audit Log Filtering”.

### 6.7 MySQL Enterprise Firewall

Note

MySQL Enterprise Firewall is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see [https://www.mysql.com/products/](https://www.mysql.com/products/).

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

Each MySQL account registered with the firewall has its own statement whitelist, enabling protection to be tailored per account. For a given account, the firewall can operate in recording, protecting, or detecting mode, for training in the accepted statement patterns, active protection against unacceptable statements, or passive detection of unacceptable statements. The diagram illustrates how the firewall processes incoming statements in each mode.
The following sections describe the components of MySQL Enterprise Firewall, discuss how to install and use it, and provide reference information for its components.

6.7.1 MySQL Enterprise Firewall Components

MySQL Enterprise Firewall is based on a plugin library that implements these components:

- A server-side plugin named `MYSQL_FIREWALL` examines SQL statements before they execute and, based on its in-memory cache, renders a decision whether to execute or reject each statement.

- Server-side plugins named `MYSQL_FIREWALL_USERS` and `MYSQL_FIREWALL_WHITELIST` implement `INFORMATION_SCHEMA` tables that provide views into the firewall data cache.

- System tables named `firewall_users` and `firewall_whitelist` in the `mysql` database provide persistent storage of firewall data.

- Stored procedures named `sp_set_firewall_mode()` and `sp_reload_firewall_rules()` perform tasks such as registering MySQL accounts with the firewall, establishing their operational mode, and managing transfer of firewall data between the cache and the underlying system tables.

- A set of user-defined functions provides an SQL-level API for lower-level tasks such as synchronizing the cache with the underlying system tables.

- System variables enable firewall configuration and status variables provide runtime operational information.

- `FIREWALL_ADMIN` and `FIREWALL_USER` privileges enable users to administer firewall rules for any user, and their own firewall rules, respectively.

6.7.2 Installing or Uninstalling MySQL Enterprise Firewall

MySQL Enterprise Firewall installation is a one-time operation that installs the components described in Section 6.7.1, “MySQL Enterprise Firewall Components”. Installation can be performed using a graphical interface or manually:
Installing or Uninstalling MySQL Enterprise Firewall

- On Windows, MySQL Installer includes an option to enable MySQL Enterprise Firewall for you.

- MySQL Workbench 6.3.4 or higher can install MySQL Enterprise Firewall, enable or disable an installed firewall, or uninstall the firewall.

- Manual MySQL Enterprise Firewall installation involves running a script located in the `share` directory of your MySQL installation.

Important

Read this entire section before following its instructions. Parts of the procedure differ depending on your environment.

Note

If installed, MySQL Enterprise Firewall involves some minimal overhead even when disabled. To avoid this overhead, do not install the firewall unless you plan to use it.

For usage instructions, see Section 6.7.3, “Using MySQL Enterprise Firewall”. For reference information, see Section 6.7.4, “MySQL Enterprise Firewall Reference”.

- Installing MySQL Enterprise Firewall
- Uninstalling MySQL Enterprise Firewall

Installing MySQL Enterprise Firewall

If MySQL Enterprise Firewall is already installed from an older version of MySQL, uninstall it using the instructions given later in this section and then restart your server before installing the current version. In this case, it is also necessary to register your configuration again.

On Windows, you can use MySQL Installer to install MySQL Enterprise Firewall, as shown in Figure 6.2, “MySQL Enterprise Firewall Installation on Windows”. Check the Enable MySQL Enterprise Firewall check box. (Open Firewall port for network access has a different purpose. It refers to Windows Firewall and controls whether Windows blocks the TCP/IP port on which the MySQL server listens for client connections.)

Important

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

The workaround is to run MySQL Installer without MySQL Enterprise Firewall selected. (That is, do not select the Enable MySQL Enterprise Firewall check box.) Then install MySQL Enterprise Firewall afterward using the instructions for manual installation later in this section. This problem is corrected in MySQL 8.0.20.
To install MySQL Enterprise Firewall using MySQL Workbench 6.3.4 or higher, see MySQL Enterprise Firewall Interface.

To install MySQL Enterprise Firewall manually, look in the `share` directory of your MySQL installation and choose the script that is appropriate for your platform. The available scripts differ in the suffix used to refer to the plugin library file:

- `win_install_firewall.sql`: Choose this script for Windows systems that use `.dll` as the file name suffix.
- `linux_install_firewall.sql`: Choose this script for Linux and similar systems that use `.so` as the file name suffix.

The installation script creates stored procedures in the default database, so choose a database to use. Then run the script as follows, naming the chosen database on the command line. The example here uses the `mysql` database and the Linux installation script. Make the appropriate substitutions for your system.

```
shell> mysql -u root -p mysql < linux_install_firewall.sql
Enter password: (enter root password here)
```

**Note**

To use MySQL Enterprise Firewall in the context of master/slave replication, Group Replication, or InnoDB cluster, you must prepare the slave or secondary nodes prior to running the installation script on the master or primary node. This is necessary because the `INSTALL PLUGIN` statements in the script are not replicated.
Using MySQL Enterprise Firewall

1. On each slave or secondary node, extract the `INSTALL PLUGIN` statements from the installation script and execute them manually.

2. On the master or primary node, run the installation script as described previously.

Installing MySQL Enterprise Firewall either using a graphical interface or manually should enable the firewall. To verify that, connect to the server and execute this statement:

```sql
mysql> SHOW GLOBAL VARIABLES LIKE 'mysql_firewall_mode';
+---------------------+-------+
| Variable_name       | Value |
+---------------------+-------+
| mysql_firewall_mode | ON    |
+---------------------+-------+
```

If the plugin fails to initialize, check the server error log for diagnostic messages.

Uninstalling MySQL Enterprise Firewall

MySQL Enterprise Firewall can be uninstalled using MySQL Workbench or manually.

To uninstall MySQL Enterprise Firewall using MySQL Workbench 6.3.4 or higher, see MySQL Enterprise Firewall Interface.

To uninstall MySQL Enterprise Firewall manually, execute the following statements. It is assumed that the stored procedures were created in the `mysql` database. Adjust the `DROP PROCEDURE` statements appropriately if the procedures were created in a different database.

```sql
DROP TABLE mysql.firewall_whitelist;
DROP TABLE mysql.firewall_users;
UNINSTALL PLUGIN mysql_firewall;
UNINSTALL PLUGIN mysql_firewall_whitelist;
UNINSTALL PLUGIN mysql_firewall_users;
DROP FUNCTION set_firewall_mode;
DROP FUNCTION normalize_statement;
DROP FUNCTION read_firewall_whitelist;
DROP FUNCTION read_firewall_users;
DROP FUNCTION mysql_firewall_flush_status;
DROP PROCEDURE mysql.sp_set_firewall_mode;
DROP PROCEDURE mysql.sp_reload_firewall_rules;
```

6.7.3 Using MySQL Enterprise Firewall

Before using MySQL Enterprise Firewall, install it according to the instructions provided in Section 6.7.2, "Installing or Uninstalling MySQL Enterprise Firewall".

This section describes how to configure MySQL Enterprise Firewall using SQL statements. Alternatively, MySQL Workbench 6.3.4 or higher provides a graphical interface for firewall control. See MySQL Enterprise Firewall Interface.

To enable or disable the firewall, set the `mysql_firewall_mode` system variable. By default, this variable is enabled when the firewall is installed. To control the initial firewall state explicitly, you can set the variable at server startup. For example, to enable the firewall in an option file, use these lines:

```
[mysqld]
mysql_firewall_mode=ON
```

It is also possible to disable or enable the firewall at runtime:

```sql
mysql> SET GLOBAL mysql_firewall_mode = OFF;
mysql> SET GLOBAL mysql_firewall_mode = ON;
```

In addition to the global on/off firewall mode, each account registered with the firewall has its own operational mode. For an account in recording mode, the firewall learns an application’s “fingerprint,” that is, the acceptable statement patterns that, taken together, form a whitelist. After training, switch
the firewall to protecting mode to harden MySQL against access by statements that deviate from
the fingerprint. For additional training, switch the firewall back to recording mode as necessary to
update the whitelist with new statement patterns. An intrusion-detection mode is available that writes
suspicious statements to the error log but does not deny access.

The firewall maintains whitelist rules on a per-account basis, enabling implementation of protection
strategies such as these:

• For an application that has unique protection requirements, configure it to use an account that is not
  used for any other purpose.

• For applications that are related and share protection requirements, configure them as a group to
  use the same account.

Firewall operation is based on conversion of SQL statements to normalized digest form. Firewall
digests are like the statement digests used by the Performance Schema (see Performance Schema
Statement Digests and Sampling). However, unlike the Performance Schema, the relevant digest-
related system variable is \texttt{max_digest_length}.

For a connection from a registered account, the firewall converts each incoming statement to
normalized form and processes it according to the account mode:

• In recording mode, the firewall adds the normalized statement to the account whitelist rules.

• In protecting mode, the firewall compares the normalized statement to the account whitelist rules. If
  there is a match, the statement passes and the server continues to process it. Otherwise, the server
  rejects the statement and returns an error to the client. The firewall also writes the rejected statement
to the error log if the \texttt{mysql_firewall_trace} system variable is enabled.

• In detecting mode, the firewall matches statements as in protecting mode, but writes nonmatching
  statements to the error log without denying access.

Accounts that have a mode of \texttt{OFF} or are not registered with the firewall are ignored by it.

To protect an account using MySQL Enterprise Firewall, follow these steps:

1. Register the account and put it in recording mode.

2. Connect to the MySQL server using the registered account and execute statements to be learned.
   This establishes the account’s whitelist of accepted statements.

3. Switch the registered account to protecting mode.

The following example shows how to register an account with the firewall, use the firewall to learn
acceptable statements for that account, and protect the account against execution of unacceptable
statements. The example account, \texttt{fwuser@localhost}, is for use by an application that
accesses tables in the \texttt{sakila} database. (This database is available at \url{https://dev.mysql.com/doc/index-other.html}.)

\textbf{Note}

The user and host parts of the account name are quoted separately for statements such as \texttt{CREATE USER} and \texttt{GRANT}, whereas to specify an
account for use with a firewall component, name it as a single quoted string
\texttt{fwuser@localhost}.

The convention for naming accounts as a single quoted string for firewall
components means that you cannot use accounts that have embedded \texttt{@}
characters in the user name.

Perform the steps in the following procedure using an administrative MySQL account, except those
designated for execution by the account registered with the firewall. The default database should be
\texttt{sakila} for statements executed using the registered account.
1. If necessary, create the account to be protected (choose an appropriate password) and grant it privileges for the sakila database:

```sql
mysql> CREATE USER 'fwuser'@'localhost' IDENTIFIED BY 'fWp@3sw0rd';
mysql> GRANT ALL ON sakila.* TO 'fwuser'@'localhost';
```

2. Use the `sp_set_firewall_mode()` stored procedure to register the account with the firewall and place it in recording mode (if the procedure is located in a database other than mysql, adjust the statement accordingly):

```sql
mysql> CALL mysql.sp_set_firewall_mode('fwuser@localhost', 'RECORDING');
```

During the course of its execution, the stored procedure invokes firewall user-defined functions, which may produce output of their own.

3. Using the registered account, connect to the server, then execute some statements that are legitimate for it:

```sql
mysql> SELECT first_name, last_name FROM customer WHERE customer_id = 1;
mysql> UPDATE rental SET return_date = NOW() WHERE rental_id = 1;
mysql> SELECT get_customer_balance(1, NOW());
```

The firewall converts the statements to digest form and records them in the account whitelist.

**Note**

Until the account executes statements in recording mode, its whitelist is empty, which is equivalent to “deny all.” If switched to protecting mode, the account will be effectively prohibited from executing statements.

4. At this point, the user and whitelist information is cached and can be seen in the firewall `INFORMATION_SCHEMA` tables:

```sql
mysql> SELECT MODE FROM INFORMATION_SCHEMA.MYSQL_FIREWALL_USERS
WHERE USERHOST = 'fwuser@localhost';
```

+-----------+
| MODE      |
+-----------+
| RECORDING |
+-----------+

```sql
mysql> SELECT RULE FROM INFORMATION_SCHEMA.MYSQL_FIREWALL_WHITELIST
WHERE USERHOST = 'fwuser@localhost';
```

+----------------------------------------------------------------------------+
<table>
<thead>
<tr>
<th>RULE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT <code>first_name</code> , <code>last_name</code> FROM <code>customer</code> WHERE <code>customer_id</code> = ?</td>
</tr>
<tr>
<td>SELECT <code>get_customer_balance</code> ( ? , NOW ( ) )</td>
</tr>
<tr>
<td>UPDATE <code>rental</code> SET <code>return_date</code> = NOW ( ) WHERE <code>rental_id</code> = ?</td>
</tr>
<tr>
<td>SELECT @@'version_comment' LIMIT ?</td>
</tr>
<tr>
<td>----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>

**Note**

The `@@version_comment` rule comes from a statement sent automatically by the `mysql` client when you connect to the server as the registered user.

It is important to train the firewall under conditions matching application use. For example, a given MySQL connector might send statements to the server at the beginning of a connection to determine server characteristics and capabilities. If an application normally is used through that connector, train the firewall that way, too. That enables those initial statements to become part of the whitelist for the account associated with the application.

5. Use the stored procedure to switch the registered user to protecting mode:

```sql
mysql> CALL mysql.sp_set_firewall_mode('fwuser@localhost', 'PROTECTING');
```
Using MySQL Enterprise Firewall

Important

Switching the account out of `RECORDING` mode synchronizes its firewall cache data to the underlying `mysql` system database tables for persistent storage. If you do not switch the mode for a user who is being recorded, the cached whitelist data is not written to the system tables and will be lost when the server is restarted.

6. Using the registered account, execute some acceptable and unacceptable statements. The firewall matches each one against the account whitelist and accepts or rejects it.

This statement is not identical to a training statement but produces the same normalized statement as one of them, so the firewall accepts it:

```
mysql> SELECT first_name, last_name FROM customer WHERE customer_id = '48';
+------------+-----------+
| first_name | last_name |
| ANN        | EVANS     |
+------------+-----------+
```

These statements do not match anything in the whitelist and each results in an error:

```
mysql> SELECT first_name, last_name FROM customer WHERE customer_id = 1 OR TRUE;
ERROR 1045 (28000): Statement was blocked by Firewall
mysql> SHOW TABLES LIKE 'customer%';
ERROR 1045 (28000): Statement was blocked by Firewall
mysql> TRUNCATE TABLE mysql.slow_log;
ERROR 1045 (28000): Statement was blocked by Firewall
```

The firewall also writes the rejected statements to the error log if the `mysql_firewall_trace` system variable is enabled. For example:

```
[Note] Plugin MYSQL_FIREWALL reported:
'ACCESS DENIED for fwuser@localhost. Reason: No match in whitelist. Statement: TRUNCATE TABLE `mysql` . `slow_log` ' 
```

You can use these log messages in your efforts to identify the source of attacks.

7. You can log nonmatching statements as suspicious without denying access. To do this, put the account in intrusion-detecting mode:

```
mysql> CALL mysql.sp_set_firewall_mode('fwuser@localhost', 'DETECTING');
```

8. Using the registered account, connect to the server, then execute a statement that does not match the whitelist:

```
mysql> SHOW TABLES LIKE 'customer%';
+------------------------------+
| Tables_in_sakila (customer%) |
| customer                     |
| customer_list                |
+------------------------------+
```

In detecting mode, the firewall permits the nonmatching statement to execute but writes a message to the error log:

```
[Note] Plugin MYSQL_FIREWALL reported:
'SUSPICIOUS STATEMENT from 'fwuser@localhost'. Reason: No match in whitelist.
```
Statement: SHOW TABLES LIKE ' ? ' 

Note
Detection mode writes messages as Notes, which are information messages. To ensure that such messages appear in the error log and are not discarded, make sure that the log_error_verbosity system variable is set to a value of 3.

9. To assess firewall activity, examine its status variables:

```
mysql> SHOW GLOBAL STATUS LIKE 'Firewall%';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Firewall_access_denied     | 3     |
| Firewall_access_granted    | 4     |
| Firewall_access_suspicious | 1     |
| Firewall_cached_entries    | 4     |
+----------------------------+-------+
```

The variables indicate the number of statements rejected, accepted, logged as suspicious, and added to the cache, respectively. The Firewall_access_granted count is 4 because of the @@version_comment statement sent by the mysql client each of the three time you used it to connect as the registered user, plus the SHOW TABLES statement that was not blocked in DETECTING mode.

Should additional training for an account be necessary, switch it to recording mode again, then back to protecting mode after executing statements to be added to the whitelist.

6.7.4 MySQL Enterprise Firewall Reference

The following discussion serves as a reference to MySQL Enterprise Firewall components:

- MySQL Enterprise Firewall Tables
- MySQL Enterprise Firewall Procedures and Functions
- MySQL Enterprise Firewall System Variables
- MySQL Enterprise Firewall Status Variables

MySQL Enterprise Firewall Tables

MySQL Enterprise Firewall maintains account and whitelist information. It uses tables in the mysql system database to store this data in persistent form, and INFORMATION_SCHEMA tables to provide views into cached data. When enabled, the firewall bases its operational decisions on the cached data.

The mysql tables can be accessed only by users with privileges for that database. The INFORMATION_SCHEMA tables are accessible by anyone.

The mysql.firewall_users table lists registered firewall accounts and their operational modes. The table has the following columns (with the corresponding INFORMATION_SCHEMA.MYSQL_FIREWALL_USERS table having similar but not necessarily identical columns):

- USERHOST

An account registered with the firewall. Each account has the format user_name@host_name and represents actual user and host names as authenticated by the server. Patterns and netmasks should not be used when registering users.

- MODE
The current firewall operational mode for the account. The permitted mode values are OFF, DETECTING, PROTECTING, RECORDING, and RESET. For details about their meanings, see the description of sp_set_firewall_mode() in MySQL Enterprise Firewall Procedures and Functions.

The mysql.firewall_whitelist table lists registered firewall accounts and their whitelists. The table has the following columns (with the corresponding INFORMATION_SCHEMA.MYSQL_FIREWALL_WHITELIST table having similar but not necessarily identical columns):

- USERHOST
  An account registered with the firewall. The format is the same as for the user account tables.

- RULE
  A normalized statement indicating an acceptable statement pattern for the account. An account whitelist is the union of its rules.

- ID
  An integer column that is a primary key for the table. This column was added in MySQL 8.0.12.

MySQL Enterprise Firewall Procedures and Functions

MySQL Enterprise Firewall has stored procedures that perform tasks such as registering MySQL accounts with the firewall, establishing their operational mode, and managing transfer of firewall data between the cache and the underlying system tables. It also has a set of user-defined functions (UDFs) that provides an SQL-level API for lower-level tasks such as synchronizing the cache with the underlying system tables.

Under normal operation, the stored procedures implement the user interface. The UDFs are invoked by the stored procedures, not directly by users.

To invoke a stored procedure when the default database is not the database that contains the procedure, qualify the procedure name with the database name. For example:

```
CALL mysql.sp_set_firewall_mode(user, mode);
```

The following list describes each firewall stored procedure and UDF:

- sp_reload_firewall_rules(user)
  This stored procedure uses firewall UDFs to reset a registered account and reload the in-memory rules for it from the rules stored in the mysql.firewall_whitelist table. This procedure provides control over firewall operation for individual accounts.

  The user argument names the affected account, as a string in user_name@host_name format.

  Example:

  ```
  CALL mysql.sp_reload_firewall_rules('fwuser@localhost');
  ```

  Warning
  This procedure sets the account mode to RESET, which clears the account whitelist and sets its mode to OFF. If the account mode was not OFF prior to the sp_reload_firewall_rules() call, use sp_set_firewall_mode() to restore its previous mode after reloading the rules. For example, if the account was in PROTECTING mode, that is no longer true after calling sp_reload_firewall_rules() and you must set it to PROTECTING again explicitly.
• **sp_set_firewall_mode**(user, mode)

This stored procedure registers a MySQL account with the firewall and establishes its operational mode. The procedure also invokes firewall UDFs as necessary to transfer firewall data between the cache and the underlying system tables. This procedure may be called even if the `mysql_firewall_mode` system variable is `OFF`, although setting the mode for an account has no operational effect while the firewall is disabled.

The `user` argument names the affected account, as a string in `user_name@host_name` format.

The `mode` is the operational mode for the user, as a string. These mode values are permitted:

- **OFF**: Disable the firewall for the account.
- **DETECTING**: Intrusion-detection mode: Write suspicious (nonmatching) statements to the error log but do not deny access.
- **PROTECTING**: Protect the account by matching incoming statements against the account whitelist.
- **RECORDING**: Training mode: Record acceptable statements for the account. Incoming statements that do not immediately fail with a syntax error are recorded to become part of the account whitelist rules.
- **RESET**: Clear the account whitelist and set the account mode to **OFF**.

Switching the mode for an account to any mode but **RECORDING** synchronizes the firewall cache data to the underlying `mysql` system database tables for persistent storage. Switching the mode from **OFF** to **RECORDING** reloads the whitelist from the `mysql.firewall_whitelist` table into the cache.

If an account has an empty whitelist, setting its mode to **PROTECTING** produces an error message that is returned in a result set, but not an SQL error:

```sql
mysql> CALL mysql.sp_set_firewall_mode('a@b','PROTECTING');
+----------------------------------------------------------------------+
| set_firewall_mode(arg_userhost, arg_mode)                            |
| ERROR: PROTECTING mode requested for a@b but the whitelist is empty. |
+----------------------------------------------------------------------+
1 row in set (0.02 sec)
Query OK, 0 rows affected (0.02 sec)
```

• **mysql_firewall_flush_status()**

This UDF resets several firewall status variables to 0:

```
Firewall_access_denied
Firewall_access_granted
Firewall_access_suspicious
```

Example:

```sql
SELECT mysql_firewall_flush_status();
```

• **normalize_statement**(stmt)

This UDF normalizes an SQL statement into the digest form used for whitelist rules.

Example:

```sql
SELECT normalize_statement('SELECT * FROM t1 WHERE c1 > 2');
```
MySQL Enterprise Firewall Reference

- **read_firewall_users(user, mode)**
  
  This aggregate UDF updates the firewall user cache through a `SELECT` statement on the `mysql.firewall_users` table.

  Example:
  ```sql
  SELECT read_firewall_users('fwuser@localhost', 'RECORDING')
  FROM mysql.firewall_users;
  ```

- **read_firewall_whitelist(user, rule)**
  
  This aggregate UDF updates the recorded statement cache through a `SELECT` statement on the `mysql.firewall_whitelist` table.

  Example:
  ```sql
  SELECT read_firewall_whitelist('fwuser@localhost', 'RECORDING')
  FROM mysql.firewall_whitelist;
  ```

- **set_firewall_mode(user, mode)**
  
  This UDF manages the user cache and establishes the user operational mode.

  Example:
  ```sql
  SELECT set_firewall_mode('fwuser@localhost', 'RECORDING');
  ```

### MySQL Enterprise Firewall System Variables

MySQL Enterprise Firewall supports the following system variables. Use them to configure firewall operation. These variables are unavailable unless the firewall is installed (see Section 6.7.2, “Installing or Uninstalling MySQL Enterprise Firewall”).

- **mysql_firewall_mode**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td>`--mysql-firewall-mode=[OFF</td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>mysql_firewall_mode</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><code>SET_VAR</code> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>ON</td>
</tr>
</tbody>
</table>

Whether MySQL Enterprise Firewall is enabled (the default) or disabled.

- **mysql_firewall_trace**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Format</td>
<td>`--mysql-firewall-trace=[OFF</td>
</tr>
<tr>
<td>Introduced</td>
<td>8.0.11</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>mysql_firewall_trace</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Whether the MySQL Enterprise Firewall trace is enabled or disabled (the default). When `mysql_firewall_trace` is enabled, for PROTECTING mode, the firewall writes rejected statements to the error log.

MySQL Enterprise Firewall Status Variables

MySQL Enterprise Firewall supports the following status variables. Use them to obtain information about firewall operational status. These variables are unavailable unless the firewall is installed (see Section 6.7.2, "Installing or Uninstalling MySQL Enterprise Firewall"). Firewall status variables are set to 0 whenever the `MYSQL_FIREWALL` plugin is installed or the server is started. Many of them are reset to zero by the `mysql_firewall_flush_status()` UDF (see MySQL Enterprise Firewall Procedures and Functions).

- **Firewall_access_denied**
  The number of statements rejected by MySQL Enterprise Firewall.

- **Firewall_access_granted**
  The number of statements accepted by MySQL Enterprise Firewall.

- **Firewall_access_suspicious**
  The number of statements logged by MySQL Enterprise Firewall as suspicious for users who are in DETECTING mode.

- **Firewall_cached_entries**
  The number of statements recorded by MySQL Enterprise Firewall, including duplicates.

6.8 MySQL Enterprise Data Masking and De-Identification

MySQL Enterprise Data Masking and De-Identification is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, [https://www.mysql.com/products/](https://www.mysql.com/products/).

As of MySQL 8.0.13, MySQL Enterprise Edition provides data masking and de-identification capabilities:

- Transformation of existing data to mask it and remove identifying characteristics, such as changing all digits of a credit card number but the last four to 'X' characters.

- Generation of random data, such as email addresses and payment card numbers.

The way that applications use these capabilities depends on the purpose for which the data will be used and who will access it:

- Applications that use sensitive data may protect it by performing data masking and permitting use of partially masked data for client identification. Example: A call center may ask for clients to provide their last four Social Security number digits.

- Applications that require properly formatted data, but not necessarily the original data, can synthesize sample data. Example: An application developer who is testing data validators but has no access to original data may synthesize random data with the same format.
Example 1:

Medical research facilities can hold patient data that comprises a mix of personal and medical data. This may include genetic sequences (long strings), test results stored in JSON format, and other data types. Although the data may be used mostly by automated analysis software, access to genome data or test results of particular patients is still possible. In such cases, data masking should be used to render this information not personally identifiable.

Example 2:

A credit card processor company provides a set of services using sensitive data, such as:

- Processing a large number of financial transactions per second.
- Storing a large amount of transaction-related data.
- Protecting transaction-related data with strict requirements for personal data.
- Handling client complaints about transactions using reversible or partially masked data.

A typical transaction may include many types of sensitive information, including:

- Credit card number.
- Transaction type and amount.
- Merchant type.
- Transaction cryptogram (to confirm transaction legitimacy).
- Geolocation of GPS-equipped terminal (for fraud detection).

Those types of information may then be joined within a bank or other card-issuing financial institution with client personal data, such as:

- Full client name (either person or company).
- Address.
- Date of birth.
- Social Security number.
- Email address.
- Phone number.

Various employee roles within both the card processing company and the financial institution require access to that data. Some of these roles may require access only to masked data. Other roles may require access to the original data on a case-to-case basis, which is recorded in audit logs.

Masking and de-identification are core to regulatory compliance, so MySQL Enterprise Data Masking and De-Identification can help application developers satisfy privacy requirements:

- PCI – DSS: Payment Card Data.
- HIPAA: Privacy of Health Data, Health Information Technology for Economic and Clinical Health Act (HITECH Act).
- Data Protection Act (UK): Protection of Personal Data.
The following sections describe the components of MySQL Enterprise Data Masking and De-Identification, discuss how to install and use it, and provide reference information for its components.

### 6.8.1 MySQL Enterprise Data Masking and De-Identification Components

MySQL Enterprise Data Masking and De-Identification is based on a plugin library that implements these components:

- A server-side plugin named `data_masking`.
- A set of user-defined functions (UDFs) provides an SQL-level API for performing masking and de-identification operations. Some of these functions require the `SUPER` privilege.

### 6.8.2 Installing or Uninstalling MySQL Enterprise Data Masking and De-Identification

This section describes how to install or uninstall MySQL Enterprise Data Masking and De-Identification, which is implemented as a plugin library file containing a plugin and user-defined functions (UDFs). For general information about installing or uninstalling plugins and UDFs, see [Installing and Uninstalling Plugins](#) and [Installing and Uninstalling User-Defined Functions](#).

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

The plugin library file base name is `data_masking`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

To install the MySQL Enterprise Data Masking and De-Identification plugin and UDFs, use the `INSTALL PLUGIN` and `CREATE FUNCTION` statements (adjust the `.so` suffix for your platform as necessary):

```
INSTALL PLUGIN data_masking SONAME 'data_masking.so';
CREATE FUNCTION gen_blacklist RETURNS STRING
  SONAME 'data_masking.so';
CREATE FUNCTION gen_dictionary RETURNS STRING
  SONAME 'data_masking.so';
CREATE FUNCTION gen_dictionary_drop RETURNS STRING
  SONAME 'data_masking.so';
CREATE FUNCTION gen_dictionary_load RETURNS STRING
  SONAME 'data_masking.so';
CREATE FUNCTION gen_range RETURNS INTEGER
  SONAME 'data_masking.so';
CREATE FUNCTION gen_rnd_email RETURNS STRING
  SONAME 'data_masking.so';
CREATE FUNCTION gen_rnd_pan RETURNS STRING
  SONAME 'data_masking.so';
CREATE FUNCTION gen_rnd_ssn RETURNS STRING
  SONAME 'data_masking.so';
CREATE FUNCTION gen_rnd_us_phone RETURNS STRING
  SONAME 'data_masking.so';
CREATE FUNCTION mask_inner RETURNS STRING
  SONAME 'data_masking.so';
CREATE FUNCTION mask_outer RETURNS STRING
  SONAME 'data_masking.so';
CREATE FUNCTION mask_pan RETURNS STRING
  SONAME 'data_masking.so';
CREATE FUNCTION mask_pan_relaxed RETURNS STRING
  SONAME 'data_masking.so';
CREATE FUNCTION mask_ssn RETURNS STRING
  SONAME 'data_masking.so';
```

If the plugin and UDFs are used on a master replication server, install them on all slave servers as well to avoid replication issues.
Once installed as just described, the plugin and UDFs remain installed until uninstalled. To remove them, use the `UNINSTALL PLUGIN` and `DROP FUNCTION` statements:

```sql
UNINSTALL PLUGIN data_masking;
DROP FUNCTION gen_blacklist;
DROP FUNCTION gen_dictionary;
DROP FUNCTION gen_dictionary_drop;
DROP FUNCTION gen_dictionary_load;
DROP FUNCTION gen_range;
DROP FUNCTION gen_rnd_email;
DROP FUNCTION gen_rnd_pan;
DROP FUNCTION gen_rnd_ssn;
DROP FUNCTION gen_rnd_us_phone;
DROP FUNCTION mask_inner;
DROP FUNCTION mask_outer;
DROP FUNCTION mask_pan;
DROP FUNCTION mask_pan_relaxed;
DROP FUNCTION mask_ssn;
```

### 6.8.3 Using MySQL Enterprise Data Masking and De-Identification

Before using MySQL Enterprise Data Masking and De-Identification, install it according to the instructions provided at Section 6.8.2, "Installing or Uninstalling MySQL Enterprise Data Masking and De-Identification".

To use MySQL Enterprise Data Masking and De-Identification in applications, invoke the functions that are appropriate for the operations you wish to perform. For detailed function descriptions, see Section 6.8.4, "MySQL Enterprise Data Masking and De-Identification User-Defined Function Reference". This section demonstrates how to use the functions to carry out some representative tasks. It first presents an overview of the available functions, followed by some examples of how the functions might be used in real-world context:

- Masking Data to Remove Identifying Characteristics
- Generating Random Data with Specific Characteristics
- Generating Random Data Using Dictionaries
- Using Masked Data for Customer Identification
- Creating Views that Display Masked Data

#### Masking Data to Remove Identifying Characteristics

MySQL provides general-purpose masking functions that mask arbitrary strings, and special-purpose masking functions that mask specific types of values.

**General-Purpose Masking Functions**

`mask_inner()` and `mask_outer()` are general-purpose functions that mask parts of arbitrary strings based on position within the string:

- `mask_inner()` masks the interior of its string argument, leaving the ends unmasked. Other arguments specify the sizes of the unmasked ends.

```sql
mysql> SELECT mask_inner('This is a string', 5, 1);
+--------------------------------------+
| mask_inner('This is a string', 5, 1) |
+--------------------------------------+
| This XXXXXXXXg                      |
+--------------------------------------+

mysql> SELECT mask_inner('This is a string', 1, 5);
+--------------------------------------+
| mask_inner('This is a string', 1, 5) |
+--------------------------------------+
| TXXXXXXXXXXtring                    |
+--------------------------------------+
```
• `mask_outer()` does the reverse, masking the ends of its string argument, leaving the interior unmasked. Other arguments specify the sizes of the masked ends.

```plaintext
mysql> SELECT mask_outer('This is a string', 5, 1);
+--------------------------------------+
| mask_outer('This is a string', 5, 1) |
| XXXXXis a strinX                     |
+--------------------------------------+

mysql> SELECT mask_outer('This is a string', 1, 5);
+--------------------------------------+
| mask_outer('This is a string', 1, 5) |
| Xhis is a sXXXXX                      |
+--------------------------------------+
```

By default, `mask_inner()` and `mask_outer()` use 'X' as the masking character, but permit an optional masking-character argument:

```plaintext
mysql> SELECT mask_inner('This is a string', 5, 1, '*');
+-------------------------------------------+
| mask_inner('This is a string', 5, 1, '*') |
| This **********g                          |
+-------------------------------------------+

mysql> SELECT mask_outer('This is a string', 5, 1, '#');
+-------------------------------------------+
| mask_outer('This is a string', 5, 1, '#') |
| #####is a strin#                          |
+-------------------------------------------+
```

### Special-Purpose Masking Functions

Other masking functions expect a string argument representing a specific type of value and mask it to remove identifying characteristics.

---

**Note**

The examples here supply function arguments using the random value generation functions that return the appropriate type of value. For more information about generation functions, see [Generating Random Data with Specific Characteristics](#).

---

### Payment card Primary Account Number masking.

Masking functions provide strict and relaxed masking of Primary Account Numbers.

• `mask_pan()` masks all but the last four digits of the number:

```plaintext
mysql> SELECT mask_pan(gen_rnd_pan());
+-------------------------+
| mask_pan(gen_rnd_pan()) |
| XXXXXXXXXXXX2461        |
+-------------------------+
```

• `mask_pan_relaxed()` is similar but does not mask the first six digits that indicate the payment card issuer unmasked:

```plaintext
mysql> SELECT mask_pan_relaxed(gen_rnd_pan());
+---------------------------------+
| mask_pan_relaxed(gen_rnd_pan()) |
| 770630XXXXXX0807                |
+---------------------------------+
```

### US Social Security number masking.

`mask_ssn()` masks all but the last four digits of the number:
Generating Random Data with Specific Characteristics

Several functions generate random values. These values can be used for testing, simulation, and so forth.

`gen_range()` returns a random integer selected from a given range:

```sql
mysql> SELECT gen_range(1, 10);
+------------------+
| gen_range(1, 10) |
| 6                |
+------------------+
```

`gen_rnd_email()` returns a random email address in the `example.com` domain:

```sql
mysql> SELECT gen_rnd_email();
+---------------------------+
| gen_rnd_email()           |
| ayxnq.xmkpvy@example.com  |
+---------------------------+
```

`gen_rnd_pan()` returns a random payment card Primary Account Number:

```sql
mysql> SELECT gen_rnd_pan();
```

(The `gen_rnd_pan()` function result is not shown because its return values should be used only for testing purposes, and not for publication. It cannot be guaranteed the number is not assigned to a legitimate payment account.)

`gen_rnd_ssn()` returns a random US Social Security number with the first and second parts each chosen from a range not used for legitimate numbers:

```sql
mysql> SELECT gen_rnd_ssn();
+---------------+
| gen_rnd_ssn() |
| 912-45-1615   |
+---------------+
```

`gen_rnd_us_phone()` returns a random US phone number in the 555 area code not used for legitimate numbers:

```sql
mysql> SELECT gen_rnd_us_phone();
+--------------------+
| gen_rnd_us_phone() |
| 1-555-747-5627     |
+--------------------+
```

Generating Random Data Using Dictionaries

MySQL Enterprise Data Masking and De-Identification enables dictionaries to be used as sources of random values. To use a dictionary, it must first be loaded from a file and given a name. Each loaded dictionary becomes part of the dictionary registry. Items then can be selected from registered dictionaries and used as random values or as replacements for other values.

A valid dictionary file has these characteristics:

- The file contents are plain text, one term per line.
Using MySQL Enterprise Data Masking and De-Identification

- Empty lines are ignored.
- The file must contain at least one term.

Suppose that a file named `de_cities.txt` contains these city names in Germany:

| Berlin   |
| Munich  |
| Bremen  |

Also suppose that a file named `us_cities.txt` contains these city names in the United States:

| Chicago |
| Houston |
| Phoenix |
| El Paso |
| Detroit |

Assume that the `secure_file_priv` system variable is set to `/usr/local/mysql/mysql-files`. In that case, copy the dictionary files to that directory so that the MySQL server can access them. Then use `gen_dictionary_load()` to load the dictionaries into the dictionary registry and assign them names:

```sql
mysql> SELECT gen_dictionary_load('/usr/local/mysql/mysql-files/de_cities.txt', 'DE_Cities');
+--------------------------------------------------------------------------------+
| gen_dictionary_load('/usr/local/mysql/mysql-files/de_cities.txt', 'DE_Cities') |
+--------------------------------------------------------------------------------+
| Dictionary load success                                                        |
+--------------------------------------------------------------------------------+
mysql> SELECT gen_dictionary_load('/usr/local/mysql/mysql-files/us_cities.txt', 'US_Cities');
+--------------------------------------------------------------------------------+
| gen_dictionary_load('/usr/local/mysql/mysql-files/us_cities.txt', 'US_Cities') |
+--------------------------------------------------------------------------------+
| Dictionary load success                                                        |
+--------------------------------------------------------------------------------+
```

To select a random term from a dictionary, use `gen_dictionary()`:

```sql
mysql> SELECT gen_dictionary('DE_Cities');
+-----------------------------+
| gen_dictionary('DE_Cities') |
+-----------------------------+
| Berlin                      |
+-----------------------------+
mysql> SELECT gen_dictionary('US_Cities');
+-----------------------------+
| gen_dictionary('US_Cities') |
+-----------------------------+
| Phoenix                     |
+-----------------------------+
```

To select a random term from multiple dictionaries, randomly select one of the dictionaries, then select a term from it:

```sql
mysql> SELECT gen_dictionary(ELT(gen_range(1,2), 'DE_Cities', 'US_Cities'));
+---------------------------------------------------------------+
| gen_dictionary(ELT(gen_range(1,2), 'DE_Cities', 'US_Cities')) |
+---------------------------------------------------------------+
| Detroit                                                       |
+---------------------------------------------------------------+
mysql> SELECT gen_dictionary(ELT(gen_range(1,2), 'DE_Cities', 'US_Cities'));
+---------------------------------------------------------------+
| gen_dictionary(ELT(gen_range(1,2), 'DE_Cities', 'US_Cities')) |
+---------------------------------------------------------------+
| Bremen                                                        |
+---------------------------------------------------------------+
```

The `gen_blacklist()` function enables a term from one dictionary to be replaced by a term from another dictionary, which effects masking by substitution. Its arguments are the term to replace, the
dictionary in which the term appears, and the dictionary from which to choose a replacement. For example, to substitute a US city for a German city, or vice versa, use `gen_blacklist()` like this:

```sql
mysql> SELECT gen_blacklist('Munich', 'DE_Cities', 'US_Cities');
+---------------------------------------------------+
| gen_blacklist('Munich', 'DE_Cities', 'US_Cities') |
+---------------------------------------------------+
| Houston                                           |
+---------------------------------------------------+
mysql> SELECT gen_blacklist('El Paso', 'US_Cities', 'DE_Cities');
+----------------------------------------------------+
| gen_blacklist('El Paso', 'US_Cities', 'DE_Cities') |
+----------------------------------------------------+
| Bremen                                             |
+----------------------------------------------------+
```

If the term to replace is not in the first dictionary, `gen_blacklist()` returns it unchanged:

```sql
mysql> SELECT gen_blacklist('Moscow', 'DE_Cities', 'US_Cities');
+---------------------------------------------------+
| gen_blacklist('Moscow', 'DE_Cities', 'US_Cities') |
+---------------------------------------------------+
| Moscow                                            |
+---------------------------------------------------+
```

### Using Masked Data for Customer Identification

At customer-service call centers, one common identity verification technique is to ask customers to provide their last four Social Security number (SSN) digits. For example, a customer might say her name is Joanna Bond and that her last four SSN digits are 0007.

Suppose that a `customer` table containing customer records has these columns:

- **id**: Customer ID number.
- **first_name**: Customer first name.
- **last_name**: Customer last name.
- **ssn**: Customer Social Security number.

For example, the table might be defined as follows:

```sql
CREATE TABLE customer
(
  id         BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  first_name VARCHAR(40),
  last_name  VARCHAR(40),
  ssn        VARCHAR(11)
);
```

The application used by customer-service representatives to check the customer SSN might execute a query like this:

```sql
mysql> SELECT id, ssn
FROM customer
WHERE first_name = 'Joanna' AND last_name = 'Bond';
+-----+-------------+
| id  | ssn         |
+-----+-------------+
| 786 | 906-39-0007 |
+-----+-------------+
```

However, that exposes the SSN to the customer-service representative, who has no need to see anything but the last four digits. Instead, the application can use this query to display only the masked SSN:

```sql
mysql> SELECT id, mask_ssn(CONVERT(ssn USING binary)) AS masked_ssn
FROM customer
```

---

359
MySQL Enterprise Data Masking and De-Identification User-Defined Function Reference

```
mysql> WHERE first_name = 'Joanna' AND last_name = 'Bond';
+-----+-------------+
| id  | masked_ssn  |
+-----+-------------+
| 786 | XXX-XX-0007 |
+-----+-------------+
```

Now the representative sees only what is necessary, and customer privacy is preserved.

Why was the `CONVERT()` function used for the argument to `mask_ssn()`? Because `mask_ssn()` requires an argument of length 11. Thus, even though `ssn` is defined as `VARCHAR(11)`, if the `ssn` column has a multibyte character set, it may appear to be longer than 11 bytes when passed to a UDF, and an error occurs. Converting the value to a binary string ensures that the UDF sees an argument of length 11.

A similar technique may be needed for other data masking functions when string arguments do not have a single-byte character set.

### Creating Views that Display Masked Data

If masked data from a table is used for multiple queries, it may be convenient to define a view that produces masked data. That way, applications can select from the view without performing masking in individual queries.

For example, a masking view on the `customer` table from the previous section can be defined like this:

```
CREATE VIEW masked_customer AS
SELECT id, first_name, last_name,
    mask_ssn(CONVERT(ssn USING binary)) AS masked_ssn
FROM customer;
```

Then the query to look up a customer becomes simpler but still returns masked data:

```
mysql> SELECT id, masked_ssn
mysql> FROM masked_customer
mysql> WHERE first_name = 'Joanna' AND last_name = 'Bond';
+-----+-------------+
| id  | masked_ssn  |
+-----+-------------+
| 786 | XXX-XX-0007 |
+-----+-------------+
```

### 6.8.4 MySQL Enterprise Data Masking and De-Identification User-Defined Function Reference

The MySQL Enterprise Data Masking and De-Identification plugin library includes several user-defined functions (UDFs), which may be grouped into these categories:

- **Data Masking Functions**
- **Random Data Generation Functions**
- **Random Data Dictionary-Based Functions**

As of MySQL 8.0.19, these UDFs support the single-byte `latin1` character set for string arguments and return values. Prior to MySQL 8.0.19, the UDFs treat string arguments as binary strings (which means they do not distinguish lettercase), and string return values are binary strings. You can see the difference in return value character set as follows:

MySQL 8.0.19 and higher:

```
mysql> SELECT CHARSET(gen_rnd_email());
+--------------------------+
| CHARSET(gen_rnd_email()) |
+--------------------------+
| latin1                   |
+--------------------------+
```
MySQL Enterprise Data Masking and De-Identification User-Defined Function Reference

Prior to MySQL 8.0.19:

```
mysql> SELECT CHARSET(gen_rnd_email());
+--------------------------+
| CHARSET(gen_rnd_email()) |
+--------------------------+
| binary                   |
+--------------------------+
```

For any version, if a string return value should be in a different character set, convert it. The following example shows how to convert the result of `gen_rnd_email()` to the `utf8mb4` character set:

```
SET @email = CONVERT(gen_rnd_email() USING utf8mb4);
```

It may also be necessary to convert string arguments, as illustrated in Using Masked Data for Customer Identification.

### Data Masking Functions

Each function in this section performs a masking operation on its string argument and returns the masked result.

- **mask_inner(\textit{str}, \textit{margin1}, \textit{margin2} [, \textit{mask_char}])**

  Masks the interior part of a string, leaving the ends untouched, and returns the result. An optional masking character can be specified.

  **Arguments:**

  - \textit{str}: The string to mask.
  - \textit{margin1}: A nonnegative integer that specifies the number of characters on the left end of the string to remain unmasked. If the value is 0, no left end characters remain unmasked.
  - \textit{margin2}: A nonnegative integer that specifies the number of characters on the right end of the string to remain unmasked. If the value is 0, no right end characters remain unmasked.
  - \textit{mask_char}: (Optional) The single character to use for masking. The default is 'X' if \textit{mask_char} is not given.

  The masking character must be a single-byte character. Attempts to use a multibyte character produce an error.

  **Return value:**

  The masked string, or \textbf{NULL} if either margin is negative.

  If the sum of the margin values is larger than the argument length, no masking occurs and the argument is returned unchanged.

  **Example:**

  ```
  mysql> SELECT mask_inner('abcdef', 1, 2), mask_inner('abcdef',0, 5);
  +----------------------------+---------------------------+
  | mask_inner('abcdef', 1, 2) | mask_inner('abcdef',0, 5) |
  +----------------------------+---------------------------+
  | aXXXef                     | Xbcdef                    |
  +----------------------------+---------------------------+
  mysql> SELECT mask_inner('abcdef', 1, 2, '*'), mask_inner('abcdef',0, 5, '#');
  +---------------------------------+--------------------------------+
  | mask_inner('abcdef', 1, 2, '*') | mask_inner('abcdef',0, 5, '#') |
  +---------------------------------+--------------------------------+
  | a***ef                          | #bcdef                         |
  +---------------------------------+--------------------------------+
  ```

- **mask_outer(\textit{str}, \textit{margin1}, \textit{margin2} [, \textit{mask_char}])**
Masks the left and right ends of a string, leaving the interior unmasked, and returns the result. An optional masking character can be specified.

Arguments:

- **str**: The string to mask.
- **margin1**: A nonnegative integer that specifies the number of characters on the left end of the string to mask. If the value is 0, no left end characters are masked.
- **margin2**: A nonnegative integer that specifies the number of characters on the right end of the string to mask. If the value is 0, no right end characters are masked.
- **mask_char**: (Optional) The single character to use for masking. The default is 'X' if mask_char is not given.

The masking character must be a single-byte character. Attempts to use a multibyte character produce an error.

Return value:

The masked string, or NULL if either margin is negative.

If the sum of the margin values is larger than the argument length, the entire argument is masked.

Example:

```sql
mysql> SELECT mask_outer('abcdef', 1, 2), mask_outer('abcdef', 0, 5);
+----------------------------+---------------------------+
| mask_outer('abcdef', 1, 2) | mask_outer('abcdef', 0, 5) |
| XbcdXX                    | aXXXXX                    |
+----------------------------+---------------------------+
mysql> SELECT mask_outer('abcdef', 1, 2, '*'), mask_outer('abcdef', 0, 5, '#');
+---------------------------------+--------------------------------+
| mask_outer('abcdef', 1, 2, '*') | mask_outer('abcdef', 0, 5, '#') |
| *bcd**                          | a#####                        |
+---------------------------------+--------------------------------+
```

- **mask_pan(str)**

Masks a payment card Primary Account Number and returns the number with all but the last four digits replaced by 'X' characters.

Arguments:

- **str**: The string to mask. The string must be a suitable length for the Primary Account Number, but is not otherwise checked.

Return value:

The masked payment number as a string. If the argument is shorter than required, it is returned unchanged.

Example:

```sql
mysql> SELECT mask_pan(gen_rnd_pan());
+-------------------------+
| mask_pan(gen_rnd_pan()) |
| XXXXXXXXXXXX9102        |
+-------------------------+
mysql> SELECT mask_pan(gen_rnd_pan(19));
+---------------------------------+
| mask_pan(gen_rnd_pan(19))       |
| 362                             |
+---------------------------------+
```
• **mask_pan_relaxed(str)**

Masks a payment card Primary Account Number and returns the number with all but the first six and last four digits replaced by 'X' characters. The first six digits indicate the payment card issuer.

Arguments:

• **str**: The string to mask. The string must be a suitable length for the Primary Account Number, but is not otherwise checked.

Return value:

The masked payment number as a string. If the argument is shorter than required, it is returned unchanged.

Example:

```sql
mysql> SELECT mask_pan_relaxed(gen_rnd_pan());
+---------------------------------+
| mask_pan_relaxed(gen_rnd_pan()) |
| 551279XXXXXX3108                |
+---------------------------------+

mysql> SELECT mask_pan_relaxed(gen_rnd_pan(19));
+-----------------------------------+
| mask_pan_relaxed(gen_rnd_pan(19)) |
| 462634XXXXXXXXX6739               |
+-----------------------------------+

mysql> SELECT mask_pan_relaxed('a*Z');
+-------------------------+
| mask_pan_relaxed('a*Z') |
| a*Z                     |
+-------------------------+
```

• **mask_ssn(str)**

Masks a US Social Security number and returns the number with all but the last four digits replaced by 'X' characters.

Arguments:

• **str**: The string to mask. The string must be 11 characters long, but is not otherwise checked.

Return value:

The masked Social Security number as a string, or **NULL** if the argument is not the correct length.

Example:

```sql
mysql> SELECT mask_ssn('909-63-6922'), mask_ssn('abcdefghijk');
+-------------------------+-------------------------+
| mask_ssn('909-63-6922') | mask_ssn('abcdefghijk') |
| 363                    | 363                    |
+-------------------------+-------------------------+
```
Random Data Generation Functions

The functions in this section generate random values for different types of data. When possible, generated values have characteristics reserved for demonstration or test values, to avoid having them mistaken for legitimate data. For example, `gen_rnd_us_phone()` returns a US phone number that uses the 555 area code, which is not assigned to phone numbers in actual use. Individual function descriptions describe any exceptions to this principle.

- **gen_range(lower, upper)**

  Generates a random number chosen from a specified range.

  Arguments:

  - **lower**: An integer that specifies the lower boundary of the range.
  - **upper**: An integer that specifies the upper boundary of the range, which must not be less than the lower boundary.

  Return value:

  A random integer in the range from `lower` to `upper`, inclusive, or `NULL` if the `upper` argument is less than `lower`.

  Example:

  ```
  mysql> SELECT gen_range(100, 200), gen_range(-1000, -800);
  +---------------------+------------------------+
  | gen_range(100, 200) | gen_range(-1000, -800) |
  +---------------------+------------------------+
  |                 177 |                   -917 |
  +---------------------+------------------------+
  mysql> SELECT gen_range(1, 0);
  +-----------------+
  | gen_range(1, 0) |
  +-----------------+
  |            NULL |
  +-----------------+
  ```

- **gen_rnd_email()**

  Generates a random email address in the `example.com` domain.

  Arguments:

  None.

  Return value:

  A random email address as a string.

  Example:

  ```
  mysql> SELECT gen_rnd_email();
  +---------------------------+
  | gen_rnd_email()           |
  +---------------------------+
  ```
• **gen_rnd_pan([size])**

Generates a random payment card Primary Account Number. The number passes the Luhn check (an algorithm that performs a checksum verification against a check digit).

**Warning**

Values returned from `gen_rnd_pan()` should be used only for test purposes, and are not suitable for publication. There is no way to guarantee that a given return value is not assigned to a legitimate payment account. Should it be necessary to publish a `gen_rnd_pan()` result, consider masking it with `mask_pan()` or `mask_pan_relaxed()`.

**Arguments:**

• **size**: (Optional) An integer that specifies the size of the result. The default is 16 if `size` is not given. If given, `size` must be an integer in the range from 12 to 19.

**Return value:**

A random payment number as a string, or NULL if a `size` argument outside the permitted range is given.

**Example:**

```sql
mysql> SELECT mask_pan(gen_rnd_pan());
+-------------------------+
| mask_pan(gen_rnd_pan()) |
+-------------------------+
| XXXXXXXXXXXX5805        |
+-------------------------+
mysql> SELECT mask_pan(gen_rnd_pan(19));
+---------------------------+
| mask_pan(gen_rnd_pan(19)) |
+---------------------------+
| XXXXXXXXXXXXXXX5067       |
+---------------------------+
mysql> SELECT mask_pan_relaxed(gen_rnd_pan());
+---------------------------------+
| mask_pan_relaxed(gen_rnd_pan()) |
+---------------------------------+
| 398403XXXXXX9547                |
+---------------------------------+
mysql> SELECT mask_pan_relaxed(gen_rnd_pan(19));
+-----------------------------------+
| mask_pan_relaxed(gen_rnd_pan(19)) |
+-----------------------------------+
| 578416XXXXXXXXX6509               |
+-----------------------------------+
mysql> SELECT gen_rnd_pan(11), gen_rnd_pan(20);
+-----------------+-----------------+
| gen_rnd_pan(11) | gen_rnd_pan(20) |
+-----------------+-----------------+
| NULL            | NULL            |
+-----------------+-----------------+
```
• **gen_rnd_ssn()**

Generates a random US Social Security number in *AAA-BB-CCCC* format. The *AAA* part is greater than 900 and the *BB* part is less than 70, which are characteristics not used for legitimate Social Security numbers.

Arguments:

None.

Return value:

A random Social Security number as a string.

Example:

```sql
mysql> SELECT gen_rnd_ssn();
+---------------+
| gen_rnd_ssn() |
| 951-26-0058   |
+---------------+
```

• **gen_rnd_us_phone()**

Generates a random US phone number in *1-555-BBBB* format. The 555 area code is not used for legitimate phone numbers.

Arguments:

None.

Return value:

A random US phone number as a string.

Example:

```sql
mysql> SELECT gen_rnd_us_phone();
+--------------------+
| gen_rnd_us_phone() |
| 1-555-682-5423     |
+--------------------+
```

### Random Data Dictionary-Based Functions

The functions in this section manipulate dictionaries of terms and perform generation and masking operations based on them. Some of these functions require the *SUPER* privilege.

When a dictionary is loaded, it becomes part of the dictionary registry and is assigned a name to be used by other dictionary functions. Dictionaries are loaded from plain text files containing one term per line. Empty lines are ignored. To be valid, a dictionary file must contain at least one nonempty line.

• **gen_blacklist(str, dictionary_name, replacement_dictionary_name)**

Replaces a term present in one dictionary with a term from a second dictionary and returns the replacement term. This masks the original term by substitution.

Arguments:

• *str*: A string that indicates the term to replace.

• *dictionary_name*: A string that names the dictionary containing the term to replace.
replacement_dictionary_name: A string that names the dictionary from which to choose the replacement term.

Return value:

A string randomly chosen from replacement_dictionary_name as a replacement for str, or str if it does not appear in dictionary_name, or NULL if either dictionary name is not in the dictionary registry.

If the term to replace appears in both dictionaries, it is possible for the return value to be the same term.

Example:

mysql> SELECT gen_blacklist('Berlin', 'DE_Cities', 'US_Cities');
+---------------------------------------------------+
| gen_blacklist('Berlin', 'DE_Cities', 'US_Cities') |
| Phoenix                                           |
+---------------------------------------------------+

• gen_dictionary(dictionary_name)

Returns a random term from a dictionary.

Arguments:

• dictionary_name: A string that names the dictionary from which to choose the term.

Return value:

A random term from the dictionary as a string, or NULL if the dictionary name is not in the dictionary registry.

Example:

mysql> SELECT gen_dictionary('mydict');
+--------------------------+
| gen_dictionary('mydict') |
| My term                  |
+--------------------------+

mysql> SELECT gen_dictionary('no-such-dict');
+--------------------------------+
| gen_dictionary('no-such-dict') |
| NULL                           |
+--------------------------------+
• **gen_dictionary_drop**(dictionary_name)

Removes a dictionary from the dictionary registry.

This function requires the **SUPER** privilege.

Arguments:

• *dictionary_name*: A string that names the dictionary to remove from the dictionary registry.

Return value:

A string that indicates whether the drop operation succeeded. **Dictionary removed** indicates success. **Dictionary removal error** indicates failure.

Example:

```
mysql> SELECT gen_dictionary_drop('mydict');
+-----------------------------------------------+
| gen_dictionary_drop('mydict')                 |
| Dictionary removed                            |
+-----------------------------------------------+
mysql> SELECT gen_dictionary_drop('no-such-dict');
+-----------------------------------------------+
| gen_dictionary_drop('no-such-dict')           |
| Dictionary removal error                      |
+-----------------------------------------------+
```
• `gen_dictionary_load(dictionary_path, dictionary_name)`

Loads a file into the dictionary registry and assigns the dictionary a name to be used with other functions that require a dictionary name argument.

This function requires the `SUPER` privilege.

```
<table>
<thead>
<tr>
<th>Important</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dictionaries are not persistent. Any dictionary used by applications must be loaded for each server startup.</td>
</tr>
</tbody>
</table>
```

Once loaded into the registry, a dictionary is used as is, even if the underlying dictionary file changes. To reload a dictionary, first drop it with `gen_dictionary_drop()`, then load it again with `gen_dictionary_load()`.

Arguments:

• `dictionary_path`: A string that specifies the path name of the dictionary file.

• `dictionary_name`: A string that provides a name for the dictionary.

Return value:

A string that indicates whether the load operation succeeded. `Dictionary load success` indicates success. `Dictionary load error` indicates failure. Dictionary load failure can occur for several reasons, including:

• A dictionary with the given name is already loaded.

• The dictionary file is not found.

• The dictionary file contains no terms.

• The `secure_file_priv` system variable is set and the dictionary file is not located in the directory named by the variable.

Example:

```
mysql> SELECT gen_dictionary_load('/usr/local/mysql/mysql-files/mydict','mydict');
+---------------------------------------------------------------------+
| gen_dictionary_load('/usr/local/mysql/mysql-files/mydict','mydict') |
| Dictionary load success                                            |
+---------------------------------------------------------------------+
mysql> SELECT gen_dictionary_load('/dev/null','null');
+-----------------------------------------+
| gen_dictionary_load('/dev/null','null') |
| Dictionary load error                   |
+-----------------------------------------+
```
Appendix A MySQL 8.0 FAQ: Security

Questions

• A.1: Where can I find documentation that addresses security issues for MySQL?
• A.2: What is the default authentication plugin in MySQL 8.0?
• A.3: Does MySQL 8.0 have native support for SSL?
• A.4: Is SSL support built into MySQL binaries, or must I recompile the binary myself to enable it?
• A.5: Does MySQL 8.0 have built-in authentication against LDAP directories?
• A.6: Does MySQL 8.0 include support for Roles Based Access Control (RBAC)?

Questions and Answers

A.1: Where can I find documentation that addresses security issues for MySQL?

The best place to start is Chapter 1, Security.

Other portions of the MySQL Documentation which you may find useful with regard to specific security concerns include the following:

• Section 2.1, “Security Guidelines”.
• Section 2.3, “Making MySQL Secure Against Attackers”.
• How to Reset the Root Password.
• Section 2.5, “How to Run MySQL as a Normal User”.
• UDF Security Precautions.
• Section 2.4, “Security-Related mysqld Options and Variables”.
• Section 2.6, “Security Considerations for LOAD DATA LOCAL”.
• Chapter 3, Postinstallation Setup and Testing.
• Chapter 5, Using Encrypted Connections.

A.2: What is the default authentication plugin in MySQL 8.0?

The default authentication plugin in MySQL 8.0 is caching_sha2_password. For information about this plugin, see Section 6.1.2, “Caching SHA-2 Pluggable Authentication”.

The caching_sha2_password plugin provides more secure password encryption than the mysql_native_password plugin (the default plugin in previous MySQL series). For information about the implications of this change of default plugin for server operation and compatibility of the server with clients and connectors, see caching_sha2_password as the Preferred Authentication Plugin.

For general information about pluggable authentication and other available authentication plugins, see Section 4.17, “Pluggable Authentication”, and Section 6.1, “Authentication Plugins”.

A.3: Does MySQL 8.0 have native support for SSL?

Most 8.0 binaries have support for SSL connections between the client and server. See Chapter 5, Using Encrypted Connections.
You can also tunnel a connection using SSH, if (for example) the client application does not support SSL connections. For an example, see Section 5.4, “Connecting to MySQL Remotely from Windows with SSH”.

A.4: Is SSL support built into MySQL binaries, or must I recompile the binary myself to enable it?

Most 8.0 binaries have SSL enabled for client/server connections that are secured, authenticated, or both. See Chapter 5, Using Encrypted Connections.

A.5: Does MySQL 8.0 have built-in authentication against LDAP directories?

The Enterprise edition includes a PAM Authentication Plugin that supports authentication against an LDAP directory.

A.6: Does MySQL 8.0 include support for Roles Based Access Control (RBAC)?

Not at this time.