With the server running, issue the following commands 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:

```
C:\> bin\mysqlshow
+--------------------+
|     Databases      |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
```

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

The preceding command (and commands for other MySQL programs such as `mysql`) may not work if the correct MySQL account does not exist. For example, the program may fail with an error, or you may not be able to view all databases. If you install MySQL using MySQL Installer, the `root` user is created automatically with the password you supplied. In this case, you should use the `-u root` and `-p` options. (You must use those options if you have already secured the initial MySQL accounts.) With `-p`, the client program prompts for the `root` password. For example:

```
C:\> bin\mysqlshow -u root -p
Enter password: (enter root password here)
+--------------------+
|     Databases      |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
```

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

```
C:\> bin\mysqlshow mysql
Database: mysql
+---------------------------+
|          Tables           |
+---------------------------+
| columns_priv              |
| component                 |
| db                        |
| default_roles             |
| engine_cost               |
| func                      |
| general_log               |
| global_grants             |
| gtid_executed             |
| help_category             |
| help_keyword              |
| help_relation             |
| help_topic                |
| innodb_index_stats        |
| innodb_table_stats        |
| ndb_binlog_index          |
| password_history          |
| plugin                    |
| procs_priv                |
| proxies_priv              |
| role_edges                |
| server_cost               |
| servers                   |
| slave_master_info         |
| slave_relay_log_info      |
| slave_worker_info         |
| slow_log                  |
```

```
| tables_priv               |
| time_zone                 |
| time_zone_leap_second     |
| time_zone_name            |
| time_zone_transition      |
| time_zone_transition_type |
| user                      |
+---------------------------+
```

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

```
C:\> bin\mysql -e "SELECT User, Host, plugin FROM mysql.user" mysql
+------+-----------+-----------------------+
| User | Host      | plugin                |
+------+-----------+-----------------------+
| root | localhost | caching_sha2_password |
+------+-----------+-----------------------+
```

For more information about `mysql` and `mysqlshow`, see Section 4.5.1, “mysql — The MySQL Command-Line Client”, and Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”.

2.3.7 Windows Platform Restrictions

The following restrictions apply to use of MySQL on the Windows platform:

- **Process memory**

  On Windows 32-bit platforms, it is not possible by default to use more than 2GB of RAM within a single process, including MySQL. This is because the physical address limit on Windows 32-bit is 4GB and the default setting within Windows is to split the virtual address space between kernel (2GB) and user/applications (2GB).

  Some versions of Windows have a boot time setting to enable larger applications by reducing the kernel application. Alternatively, to use more than 2GB, use a 64-bit version of Windows.

- **File system aliases**

  When using `MyISAM` tables, you cannot use aliases within Windows link to the data files on another volume and then link back to the main MySQL `datadir` location.

  This facility is often used to move the data and index files to a RAID or other fast solution.

- **Limited number of ports**

  Windows systems have about 4,000 ports available for client connections, and after a connection on a port closes, it takes two to four minutes before the port can be reused. In situations where clients connect to and disconnect from the server at a high rate, it is possible for all available ports to be used up before closed ports become available again. If this happens, the MySQL server appears to be unresponsive even though it is running. Ports may be used by other applications running on the machine as well, in which case the number of ports available to MySQL is lower.

  For more information about this problem, see https://support.microsoft.com/kb/196271.

- **`DATA DIRECTORY` and `INDEX DIRECTORY`**

  The `DATA DIRECTORY` clause of the `CREATE TABLE` statement is supported on Windows for `InnoDB` tables only, as described in Section 15.6.1.2, “Creating Tables Externally”. For `MyISAM` and other storage engines, the `DATA DIRECTORY` and `INDEX DIRECTORY` clauses for `CREATE TABLE` are ignored on Windows and any other platforms with a nonfunctional `realpath()` call.

- **`DROP DATABASE`**

  You cannot drop a database that is in use by another session.

- **Case-insensitive names**

  File names are not case-sensitive on Windows, so MySQL database and table names are also not case-sensitive on Windows. The only restriction is that database and table names must be specified using the same case throughout a given statement. See Section 9.2.3, “Identifier Case Sensitivity”.

- **Directory and file names**

  On Windows, MySQL Server supports only directory and file names that are compatible with the current ANSI code pages. For example, the following Japanese directory name does not work in the Western locale (code page 1252):

  ```
  datadir="C:/私たちのプロジェクトのデータ"
  ```

  The same limitation applies to directory and file names referred to in SQL statements, such as the data file path name in `LOAD DATA`.

- **The `\` path name separator character**

  Path name components in Windows are separated by the `\` character, which is also the escape character in MySQL. If you are using `LOAD DATA` or `SELECT ... INTO OUTFILE`, use Unix-style file names with `/` characters:

  ```
  mysql> LOAD DATA INFILE 'C:/tmp/skr.txt' INTO TABLE skr;
  mysql> SELECT * INTO OUTFILE 'C:/tmp/skr.txt' FROM skr;
  ```

  Alternatively, you must double the `\` character:

  ```
  mysql> LOAD DATA INFILE 'C:\\tmp\\skr.txt' INTO TABLE skr;
  mysql> SELECT * INTO OUTFILE 'C:\\tmp\\skr.txt' FROM skr;
  ```

- **Problems with pipes**

  Pipes do not work reliably from the Windows command-line prompt. If the pipe includes the character `^Z` / `CHAR(24)`, Windows thinks that it has encountered end-of-file and aborts the program.

  This is mainly a problem when you try to apply a binary log as follows:

  ```
  C:\> mysqlbinlog binary_log_file | mysql --user=root
  ```

  If you have a problem applying the log and suspect that it is because of a `^Z` / `CHAR(24)` character, you can use the following workaround:

  ```
  C:\> mysqlbinlog binary_log_file --result-file=/tmp/bin.sql
  C:\> mysql --user=root --execute "source /tmp/bin.sql"
  ```

  The latter command also can be used to reliably read any SQL file that may contain binary data.

2.4 Installing MySQL on macOS

For a list of macOS versions that the MySQL server supports, see https://www.mysql.com/support/supportedplatforms/database.html.

MySQL for macOS is available in a number of different forms:

- Native Package Installer, which uses the native macOS installer (DMG) to walk you through the installation of MySQL. For more information, see Section 2.4.2, “Installing MySQL on macOS Using Native Packages”. You can use the package installer with macOS. The user you use to perform the installation must have administrator privileges.
- Compressed TAR archive, which uses a file packaged using the Unix `tar` and `gzip` commands. To use this method, you need to open a `Terminal` window. You do not need administrator privileges using this method; you can install the MySQL server anywhere using this method. For

more information on using this method, you can use the generic instructions for using a tarball, Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”.

In addition to the core installation, the Package Installer also includes Section 2.4.3, “Installing and Using the MySQL Launch Daemon” and Section 2.4.4, “Installing and Using the MySQL Preference Pane” to simplify the management of your installation.

For additional information on using MySQL on macOS, see Section 2.4.1, “General Notes on Installing MySQL on macOS”.

2.4.1 General Notes on Installing MySQL on macOS

You should keep the following issues and notes in mind:

- **Other MySQL installations**: The installation procedure does not recognize MySQL installations by package managers such as Homebrew. The installation and upgrade process is for MySQL packages provided by us. If other installations are present, then consider stopping them before executing this installer to avoid port conflicts.

  Homebrew: For example, if you installed MySQL Server using Homebrew to its default location then the MySQL installer installs to a different location and won't upgrade the version from Homebrew. In this scenario you would end up with multiple MySQL installations that, by default, attempt to use the same ports. Stop the other MySQL Server instances before running this installer, such as executing *brew services stop mysql* to stop the Homebrew's MySQL service.

- **Launchd**: A launchd daemon is installed that alters MySQL configuration options. Consider editing it if needed, see the documentation below for additional information. Also, macOS 10.10 removed startup item support in favor of launchd daemons. The optional MySQL preference pane under macOS **System Preferences** uses the launchd daemon.

- **Users**: You may need (or want) to create a specific `mysql` user to own the MySQL directory and data. You can do this through the `Directory Utility`, and the `mysql` user should already exist. For use in single user mode, an entry for `_mysql` (note the underscore prefix) should already exist within the system `/etc/passwd` file.

- **Data**: Because the MySQL package installer installs the MySQL contents into a version and platform specific directory, you can use this to upgrade and migrate your database between versions. You need either to copy the `data` directory from the old version to the new version, or to specify an alternative `datadir` value to set location of the data directory. By default, the MySQL directories are installed under `/usr/local/`.

- **Aliases**: You might want to add aliases to your shell's resource file to make it easier to access commonly used programs such as `mysql` and `mysqladmin` from the command line. The syntax for `bash` is:

  ```
  alias mysql=/usr/local/mysql/bin/mysql
  alias mysqladmin=/usr/local/mysql/bin/mysqladmin
  ```

  For `tcsh`, use:

  ```
  alias mysql /usr/local/mysql/bin/mysql
  alias mysqladmin /usr/local/mysql/bin/mysqladmin
  ```

  Even better, add `/usr/local/mysql/bin` to your `PATH` environment variable. You can do this by modifying the appropriate startup file for your shell. For more information, see Section 4.2.1, “Invoking MySQL Programs”.

- **Removing**: After you have copied over the MySQL database files from the previous installation and have successfully started the new server, you should consider removing the old installation files to save disk space. Additionally, you should also remove older versions of the Package Receipt directories located in `/Library/Receipts/mysql-VERSION.pkg`.

2.4.2 Installing MySQL on macOS Using Native Packages

The package is located inside a disk image (`.dmg`) file that you first need to mount by double-clicking its icon in the Finder. It should then mount the image and display its contents.

Note

Before proceeding with the installation, be sure to stop all running MySQL server instances by using either the MySQL Manager Application (on macOS Server), the preference pane, or `mysqladmin shutdown` on the command line.

To install MySQL using the package installer:

1. Download the disk image (`.dmg`) file (the community version is available here) that contains the MySQL package installer. Double-click the file to mount the disk image and see its contents.

   Double-click the MySQL installer package from the disk. It is named according to the version of MySQL you have downloaded. For example, for MySQL server 8.0.32 it might be named `mysql-8.0.32-macos-10.13-x86_64.pkg`.

2. The initial wizard introduction screen references the MySQL server version to install. Click **Continue** to begin the installation.

   The MySQL community edition shows a copy of the relevant GNU General Public License. Click **Continue** and then **Agree** to continue.

3. From the **Installation Type** page you can either click **Install** to execute the installation wizard using all defaults, click **Customize** to alter which components to install (MySQL server, MySQL Test, Preference Pane, Launchd Support -- all but MySQL Test are enabled by default).

Note

Although the **Change Install Location** option is visible, the installation location cannot be changed.

Figure 2.13 MySQL Package Installer Wizard: Installation Type

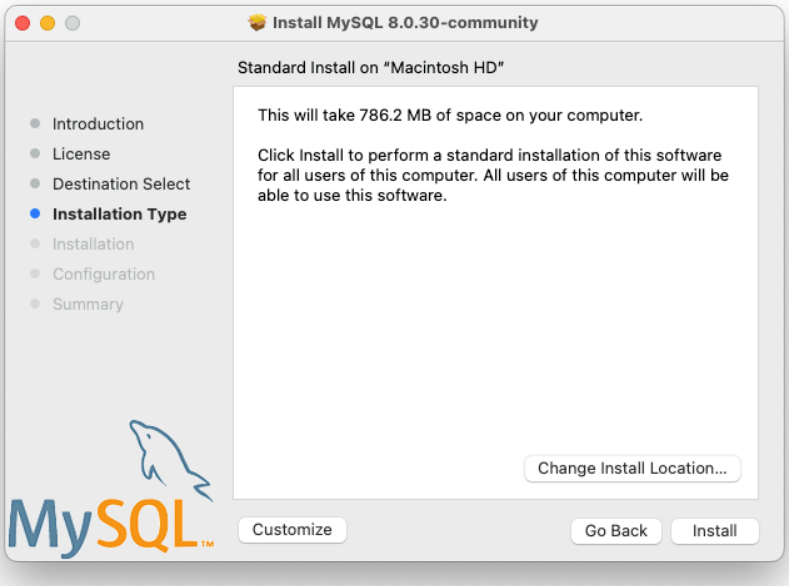
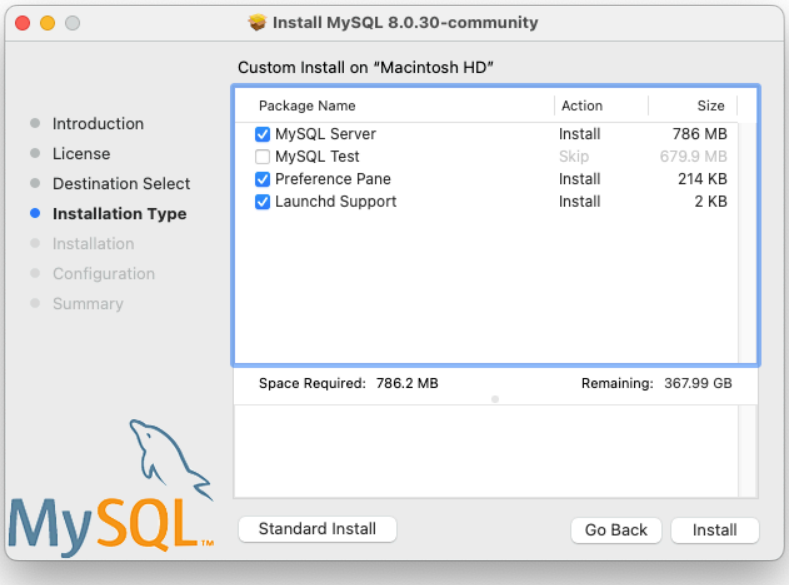


Figure 2.14 MySQL Package Installer Wizard: Customize



4. Click **Install** to install MySQL Server. The installation process ends here if upgrading a current MySQL Server installation, otherwise follow the wizard's additional configuration steps for your new MySQL Server installation.

5. After a successful new MySQL Server installation, complete the configuration steps by choosing the default encryption type for passwords, define the root password, and also enable (or disable) MySQL server at startup.

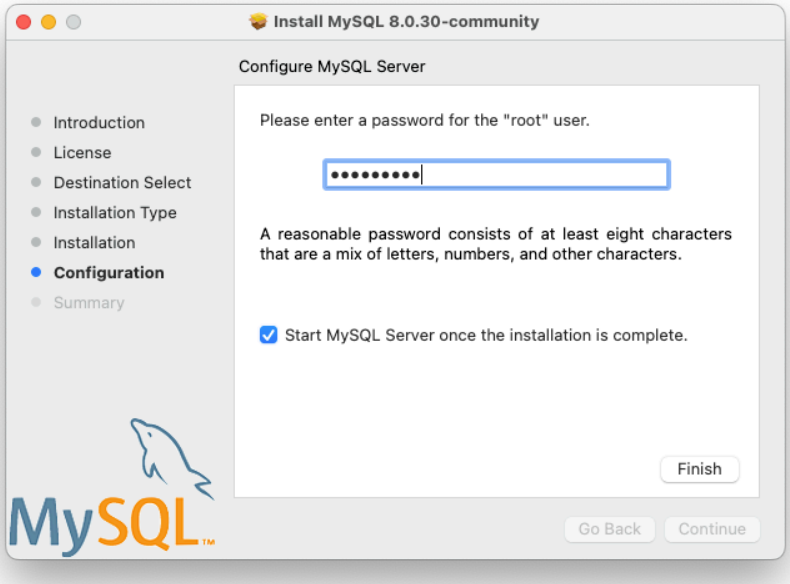
6. The default MySQL 8.0 password mechanism is `caching_sha2_password` (Strong), and this step allows you to change it to `mysql_native_password` (Legacy).

Figure 2.15 MySQL Package Installer Wizard: Choose a Password Encryption Type

Choosing the legacy password mechanism alters the generated launchd file to set `--default_authentication_plugin=mysql_native_password` under `ProgramArguments`. Choosing strong password encryption does not set `--default_authentication_plugin` because the default MySQL Server value is used, which is `caching_sha2_password`.

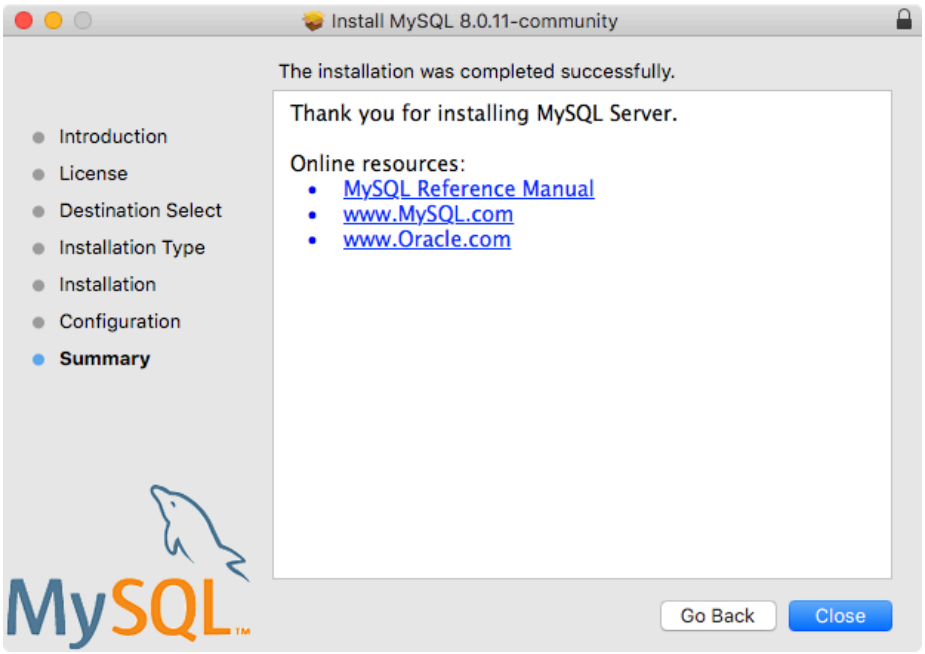
7. Define a password for the root user, and also toggle whether MySQL Server should start after the configuration step is complete.

Figure 2.16 MySQL Package Installer Wizard: Define Root Password



8. **Summary** is the final step and references a successful and complete MySQL Server installation. **Close** the wizard.

Figure 2.17 MySQL Package Installer Wizard: Summary



MySQL server is now installed. If you chose to not start MySQL, then use either launchctl from the command line or start MySQL by clicking "Start" using the MySQL preference pane. For additional information, see Section 2.4.3, “Installing and Using the MySQL Launch Daemon”, and Section 2.4.4, “Installing and Using the MySQL Preference Pane”. Use the MySQL Preference Pane or launchd to configure MySQL to automatically start at bootup.

When installing using the package installer, the files are installed into a directory within `/usr/local` matching the name of the installation version and platform. For example, the installer file `mysql-8.0.32-macos10.15-x86_64.dmg` installs MySQL into `/usr/local/mysql-8.0.32-macos10.15-x86_64/` with a symlink to `/usr/local/mysql`. The following table shows the layout of this MySQL installation directory.

Note

The macOS installation process does not create nor install a sample `my.cnf` MySQL configuration file.

Table 2.7 MySQL Installation Layout on macOS

Directory	Contents of Directory
`bin`	`mysqld` server, client and utility programs
`data`	Log files, databases, where `/usr/local/mysql/data/mysqld.local.err` is the default error log
`docs`	Helper documents, like the Release Notes and build information
`include`	Include (header) files
`lib`	Libraries
`man`	Unix manual pages
`mysql-test`	MySQL test suite ('MySQL Test' is disabled by default during the installation process when using the installer package (DMG))
`share`	Miscellaneous support files, including error messages, `dictionary.txt`, and rewriter SQL
`support-files`	Support scripts, such as `mysqld_multi.server`, `mysql.server`, and `mysql-log-rotate`.
`/tmp/mysql.sock`	Location of the MySQL Unix socket

2.4.3 Installing and Using the MySQL Launch Daemon

macOS uses launch daemons to automatically start, stop, and manage processes and applications such as MySQL.

By default, the installation package (DMG) on macOS installs a launchd file named `/Library/LaunchDaemons/com.oracle.oss.mysql.mysqld.plist` that contains a plist definition similar to:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1
<plist version="1.0">
<dict>
    <key>Label</key>            <string>com.oracle.oss.mysql.mysqld</string>
    <key>ProcessType</key>      <string>Interactive</string>
    <key>Disabled</key>         <false/>
    <key>RunAtLoad</key>        <true/>
    <key>KeepAlive</key>        <true/>
    <key>SessionCreate</key>    <true/>
    <key>LaunchOnlyOnce</key>   <false/>
    <key>UserName</key>         <string>_mysql</string>
    <key>GroupName</key>        <string>_mysql</string>
    <key>ExitTimeOut</key>      <integer>600</integer>
    <key>Program</key>          <string>/usr/local/mysql/bin/mysqld</string>
    <key>ProgramArguments</key>
        <array>
```

```
            <string>/usr/local/mysql/bin/mysqld</string>
            <string>--user=_mysql</string>
            <string>--basedir=/usr/local/mysql</string>
            <string>--datadir=/usr/local/mysql/data</string>
            <string>--plugin-dir=/usr/local/mysql/lib/plugin</string>
            <string>--log-error=/usr/local/mysql/data/mysqld.local.err</string>
            <string>--pid-file=/usr/local/mysql/data/mysqld.local.pid</string>
            <string>--keyring-file-data=/usr/local/mysql/keyring/keyring</string>
            <string>--early-plugin-load=keyring_file=keyring_file.so</string>
        </array>
    <key>WorkingDirectory</key>  <string>/usr/local/mysql</string>
</dict>
</plist>
```

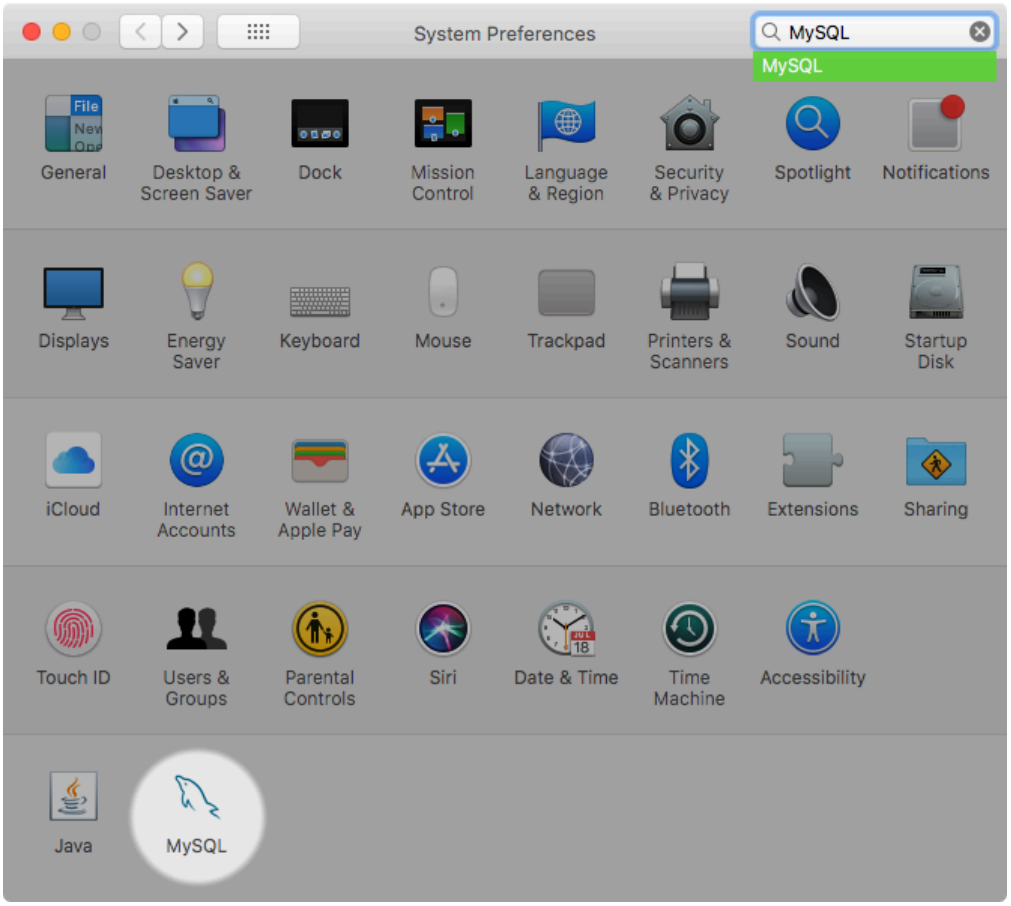
Note

Some users report that adding a plist DOCTYPE declaration causes the launchd operation to fail, despite it passing the lint check. We suspect it's a copy-n-paste error. The md5 checksum of a file containing the above snippet is *d925f05f6d1b6ee5ce5451b596d6baed*.

To enable the launchd service, you can either:

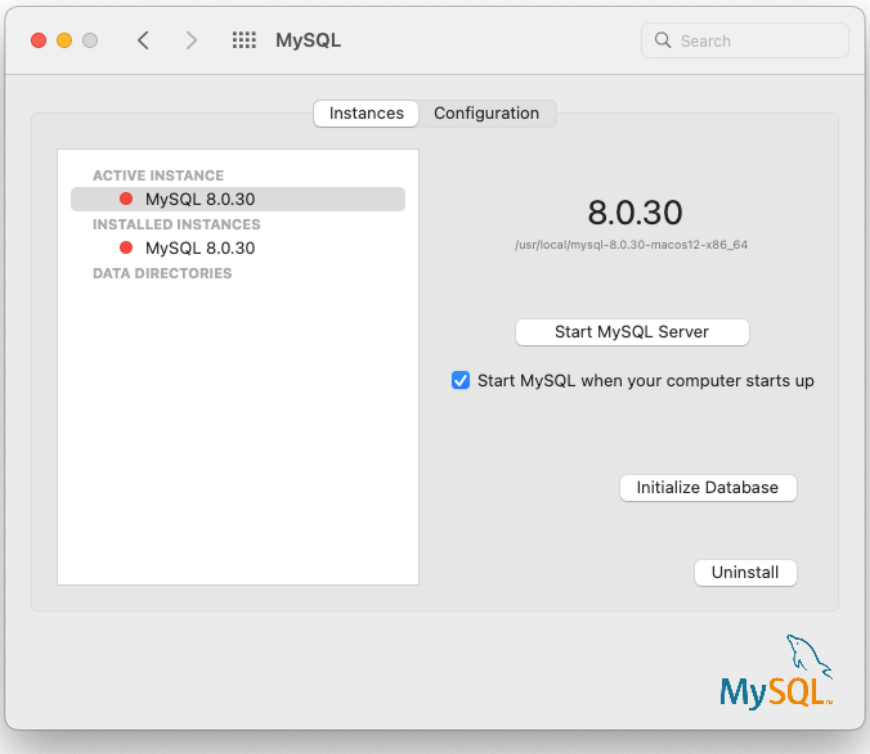
- Open macOS system preferences and select the MySQL preference panel, and then execute **Start MySQL Server**.

Figure 2.18 MySQL Preference Pane: Location



The **Instances** page includes an option to start or stop MySQL, and **Initialize Database** recreates the `data/` directory. **Uninstall** uninstalls MySQL Server and optionally the MySQL preference panel and launchd information.

Figure 2.19 MySQL Preference Pane: Instances



- Or, manually load the launchd file.

```
$> cd /Library/LaunchDaemons
$> sudo launchctl load -F com.oracle.oss.mysql.mysqld.plist
```

- To configure MySQL to automatically start at bootup, you can:

```
$> sudo launchctl load -w com.oracle.oss.mysql.mysqld.plist
```

Note

When upgrading MySQL server, the launchd installation process removes the old startup items that were installed with MySQL server 5.7.7 and below.

Upgrading also replaces your existing launchd file named `com.oracle.oss.mysql.mysqld.plist`.

Additional launchd related information:

- The plist entries override `my.cnf` entries, because they are passed in as command line arguments. For additional information about passing in program options, see Section 4.2.2, “Specifying Program Options”.
- The **ProgramArguments** section defines the command line options that are passed into the program, which is the `mysqld` binary in this case.
- The default plist definition is written with less sophisticated use cases in mind. For more complicated setups, you may want to remove some of the arguments and instead rely on a MySQL configuration file, such as `my.cnf`.

- If you edit the plist file, then uncheck the installer option when reinstalling or upgrading MySQL. Otherwise, your edited plist file is overwritten, and all edits are lost.

Because the default plist definition defines several **ProgramArguments**, you might remove most of these arguments and instead rely upon your `my.cnf` MySQL configuration file to define them. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.d
<plist version="1.0">
<dict>
    <key>Label</key>             <string>com.oracle.oss.mysql.mysqld</string>
    <key>ProcessType</key>       <string>Interactive</string>
    <key>Disabled</key>          <false/>
    <key>RunAtLoad</key>         <true/>
    <key>KeepAlive</key>         <true/>
    <key>SessionCreate</key>     <true/>
    <key>LaunchOnlyOnce</key>    <false/>
    <key>UserName</key>          <string>_mysql</string>
    <key>GroupName</key>         <string>_mysql</string>
    <key>ExitTimeOut</key>       <integer>600</integer>
    <key>Program</key>           <string>/usr/local/mysql/bin/mysqld</string>
    <key>ProgramArguments</key>
        <array>
            <string>/usr/local/mysql/bin/mysqld</string>
            <string>--user=_mysql</string>
            <string>--basedir=/usr/local/mysql</string>
            <string>--datadir=/usr/local/mysql/data</string>
            <string>--plugin-dir=/usr/local/mysql/lib/plugin</string>
            <string>--log-error=/usr/local/mysql/data/mysqld.local.err</string>
            <string>--pid-file=/usr/local/mysql/data/mysqld.local.pid</string>
            <string>--keyring-file-data=/usr/local/mysql/keyring/keyring</string>
            <string>--early-plugin-load=keyring_file=keyring_file.so</string>
        </array>
    <key>WorkingDirectory</key>  <string>/usr/local/mysql</string>
</dict>
</plist>
```

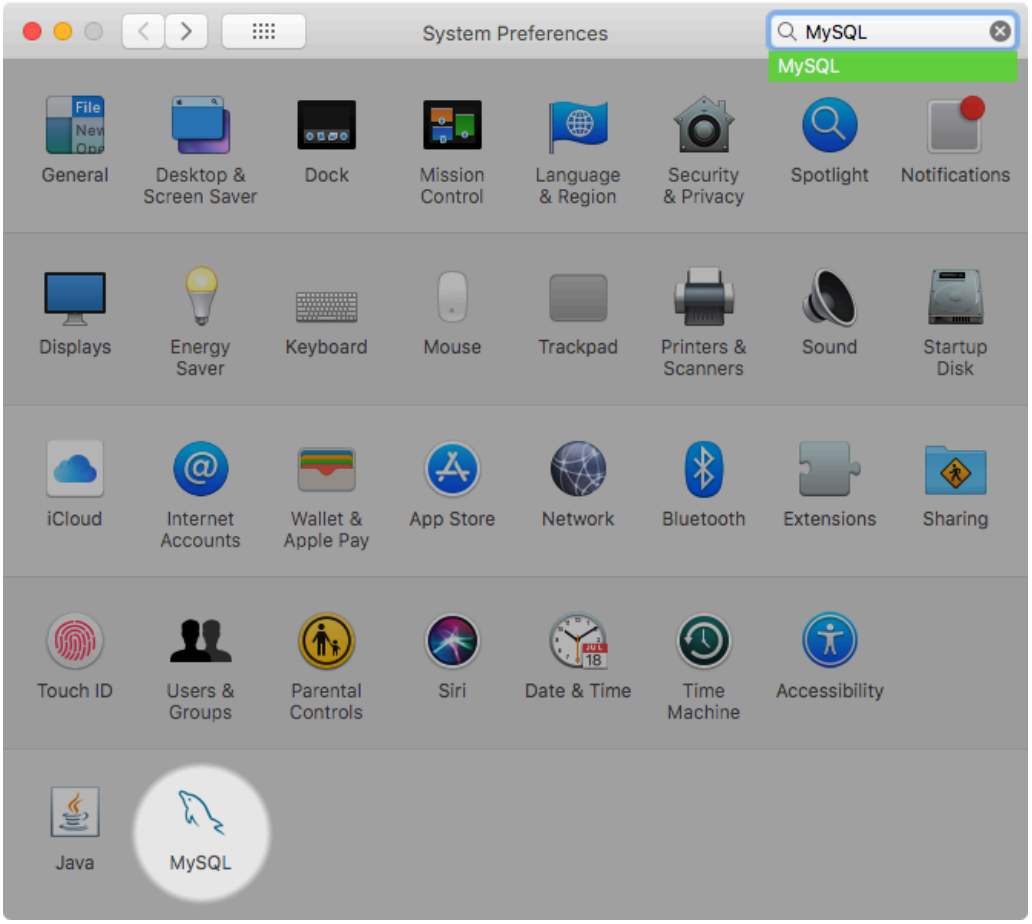
In this case, the `basedir`, `datadir`, `plugin_dir`, `log_error`, `pid_file`, `keyring_file_data`, and `--early-plugin-load` options were removed from the default plist *ProgramArguments* definition, which you might have defined in `my.cnf` instead.

2.4.4 Installing and Using the MySQL Preference Pane

The MySQL Installation Package includes a MySQL preference pane that enables you to start, stop, and control automated startup during boot of your MySQL installation.

This preference pane is installed by default, and is listed under your system's *System Preferences* window.

Figure 2.20 MySQL Preference Pane: Location



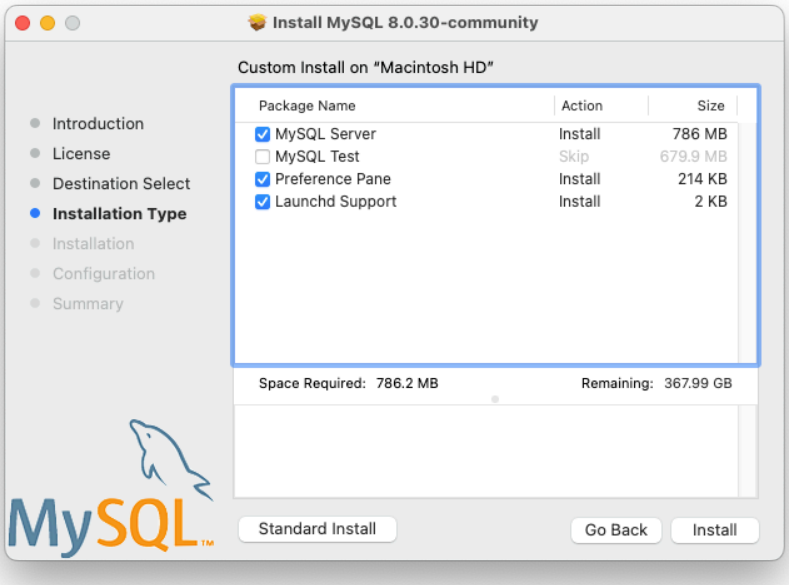
The MySQL preference pane is installed with the same DMG file that installs MySQL Server. Typically it is installed with MySQL Server but it can be installed by itself too.

To install the MySQL preference pane:

1. Go through the process of installing the MySQL server, as described in the documentation at Section 2.4.2, “Installing MySQL on macOS Using Native Packages”.

2. Click **Customize** at the **Installation Type** step. The "Preference Pane" option is listed there and enabled by default; make sure it is not deselected. The other options, such as MySQL Server, can be selected or deselected.

Figure 2.21 MySQL Package Installer Wizard: Customize



3. Complete the installation process.

Note

The MySQL preference pane only starts and stops MySQL installation installed from the MySQL package installation that have been installed in the default location.

Once the MySQL preference pane has been installed, you can control your MySQL server instance using this preference pane.

The **Instances** page includes an option to start or stop MySQL, and **Initialize Database** recreates the `data/` directory. **Uninstall** uninstalls MySQL Server and optionally the MySQL preference panel and launchd information.

Figure 2.22 MySQL Preference Pane: Instances

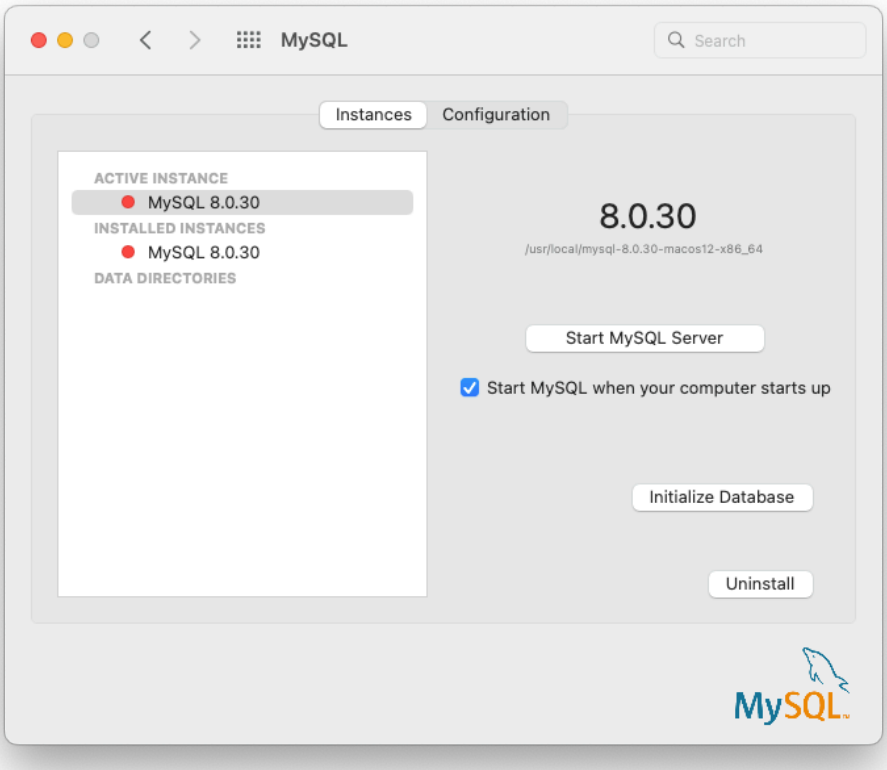
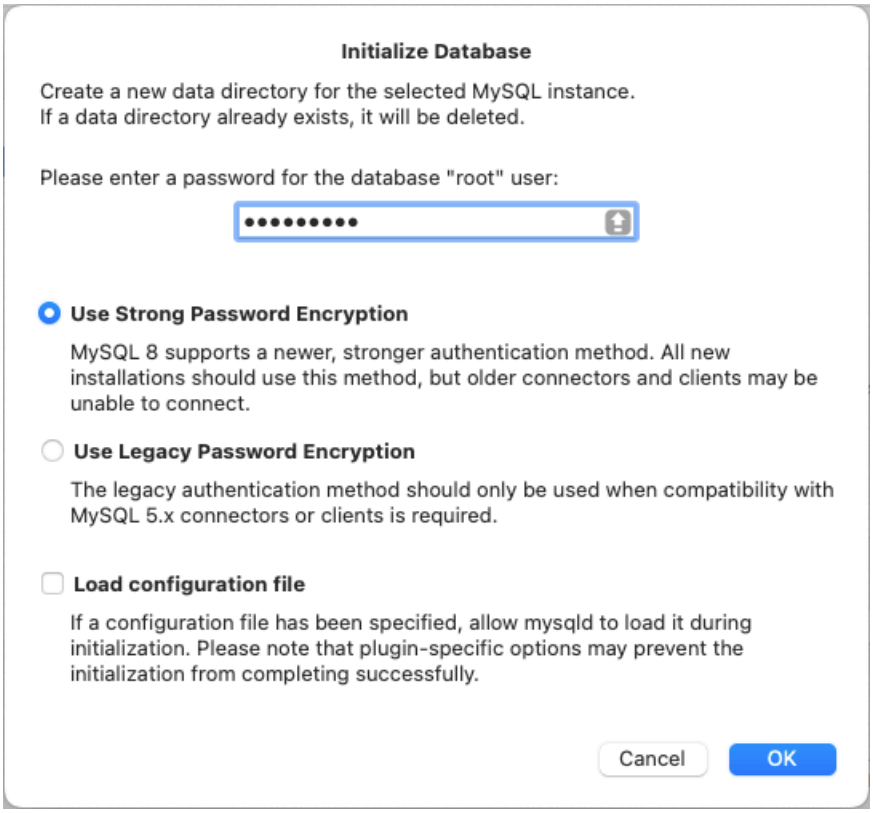
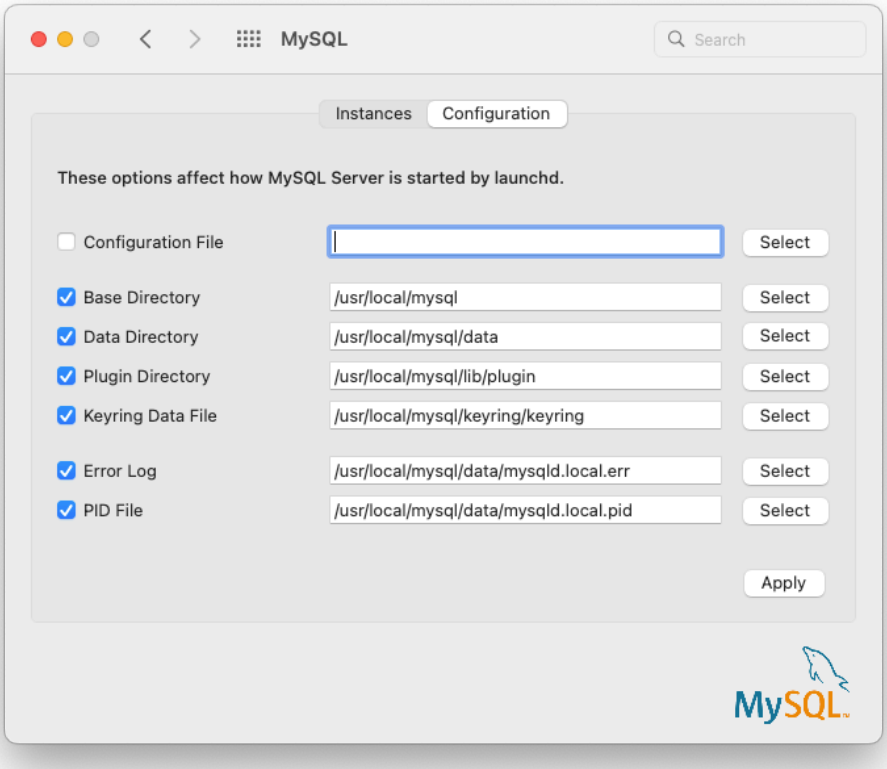


Figure 2.23 MySQL Preference Pane: Initialize Database



The **Configuration** page shows MySQL Server options including the path to the MySQL configuration file.

Figure 2.24 MySQL Preference Pane: Configuration



The MySQL Preference Pane shows the current status of the MySQL server, showing **stopped** (in red) if the server is not running and **running** (in green) if the server has already been started. The preference pane also shows the current setting for whether the MySQL server has been set to start automatically.

2.5 Installing MySQL on Linux

Linux supports a number of different solutions for installing MySQL. We recommend that you use one of the distributions from Oracle, for which several methods for installation are available:

Table 2.8 Linux Installation Methods and Information

Type	Setup Method	Additional Information
Apt	Enable the MySQL Apt repository	Documentation
Yum	Enable the MySQL Yum repository	Documentation
Zypper	Enable the MySQL SLES repository	Documentation
RPM	Download a specific package	Documentation
DEB	Download a specific package	Documentation
Generic	Download a generic package	Documentation

Type	Setup Method	Additional Information
Source	Compile from source	Documentation
Docker	Use the Oracle Container Registry. You can also use Docker Hub for MySQL Community Edition and My Oracle Support for MySQL Enterprise Edition.	Documentation
Oracle Unbreakable Linux Network	Use ULN channels	Documentation

As an alternative, you can use the package manager on your system to automatically download and install MySQL with packages from the native software repositories of your Linux distribution. These native packages are often several versions behind the currently available release. You are also normally unable to install development milestone releases (DMRs), since these are not usually made available in the native repositories. For more information on using the native package installers, see Section 2.5.7, “Installing MySQL on Linux from the Native Software Repositories”.

Note

For many Linux installations, you want to set up MySQL to be started automatically when your machine starts. Many of the native package installations perform this operation for you, but for source, binary and RPM solutions you may need to set this up separately. The required script, `mysql.server`, can be found in the `support-files` directory under the MySQL installation directory or in a MySQL source tree. You can install it as `/etc/init.d/mysql` for automatic MySQL startup and shutdown. See Section 4.3.3, “mysql.server — MySQL Server Startup Script”.

2.5.1 Installing MySQL on Linux Using the MySQL Yum Repository

The MySQL Yum repository for Oracle Linux, Red Hat Enterprise Linux, CentOS, and Fedora provides RPM packages for installing the MySQL server, client, MySQL Workbench, MySQL Utilities, MySQL Router, MySQL Shell, Connector/ODBC, Connector/Python and so on (not all packages are available for all the distributions; see Installing Additional MySQL Products and Components with Yum for details).

Before You Start

As a popular, open-source software, MySQL, in its original or re-packaged form, is widely installed on many systems from various sources, including different software download sites, software repositories, and so on. The following instructions assume that MySQL is not already installed on your system using a third-party-distributed RPM package; if that is not the case, follow the instructions given in Section 2.10.7, “Upgrading MySQL with the MySQL Yum Repository” or Replacing a Third-Party Distribution of MySQL Using the MySQL Yum Repository.

Steps for a Fresh Installation of MySQL

Follow the steps below to install the latest GA version of MySQL with the MySQL Yum repository:

Adding the MySQL Yum Repository

First, add the MySQL Yum repository to your system's repository list. This is a one-time operation, which can be performed by installing an RPM provided by MySQL. Follow these steps:

a. Go to the Download MySQL Yum Repository page (https://dev.mysql.com/downloads/repo/yum/) in the MySQL Developer Zone.

b. Select and download the release package for your platform.

c. Install the downloaded release package with the following command, replacing *platform-and-version-specific-package-name* with the name of the downloaded RPM package:

```
$> sudo yum install platform-and-version-specific-package-name.rpm
```

For an EL6-based system, the command is in the form of:

```
$> sudo yum install mysql80-community-release-el6-{version-number}.noarch.rpm
```

For an EL7-based system:

```
$> sudo yum install mysql80-community-release-el7-{version-number}.noarch.rpm
```

For an EL8-based system:

```
$> sudo yum install mysql80-community-release-el8-{version-number}.noarch.rpm
```

For an EL9-based system:

```
$> sudo yum install mysql80-community-release-el9-{version-number}.noarch.rpm
```

For Fedora 35:

```
$> sudo dnf install mysql80-community-release-fc35-{version-number}.noarch.rpm
```

For Fedora 36:

```
$> sudo dnf install mysql80-community-release-fc36-{version-number}.noarch.rpm
```

For Fedora 37:

```
$> sudo dnf install mysql80-community-release-fc37-{version-number}.noarch.rpm
```

The installation command adds the MySQL Yum repository to your system's repository list and downloads the GnuPG key to check the integrity of the software packages. See Section 2.1.4.2, “Signature Checking Using GnuPG” for details on GnuPG key checking.

You can check that the MySQL Yum repository has been successfully added by the following command (for dnf-enabled systems, replace `yum` in the command with `dnf`):

```
$> yum repolist enabled | grep "mysql.*-community.*"
```

Note

Once the MySQL Yum repository is enabled on your system, any system-wide update by the `yum update` command (or `dnf upgrade` for dnf-enabled systems) upgrades MySQL packages on your system and replaces any native third-party packages, if Yum finds replacements for them in the MySQL Yum repository; see Section 2.10.7, “Upgrading MySQL with the MySQL Yum Repository”, for a discussion on some possible effects of that on your system, see Upgrading the Shared Client Libraries.

2. Selecting a Release Series

When using the MySQL Yum repository, the latest GA series (currently MySQL 8.0) is selected for installation by default. If this is what you want, you can skip to the next step, Installing MySQL.

Within the MySQL Yum repository, different release series of the MySQL Community Server are hosted in different subrepositories. The subrepository for the latest GA series (currently MySQL 8.0) is enabled by default, and the subrepositories for all other series (for example, the MySQL 8.0 series) are disabled by default. Use this command to see all the subrepositories in the MySQL Yum

repository, and see which of them are enabled or disabled (for dnf-enabled systems, replace `yum` in the command with `dnf`):

```
$> yum repolist all | grep mysql
```

To install the latest release from the latest GA series, no configuration is needed. To install the latest release from a specific series other than the latest GA series, disable the subrepository for the latest GA series and enable the subrepository for the specific series before running the installation command. If your platform supports `yum-config-manager`, you can do that by issuing these commands, which disable the subrepository for the 5.7 series and enable the one for the 8.0 series:

```
$> sudo yum-config-manager --disable mysql57-community
$> sudo yum-config-manager --enable mysql80-community
```

For dnf-enabled platforms:

```
$> sudo dnf config-manager --disable mysql57-community
$> sudo dnf config-manager --enable mysql80-community
```

Besides using `yum-config-manager` or the `dnf config-manager` command, you can also select a release series by editing manually the `/etc/yum.repos.d/mysql-community.repo` file. This is a typical entry for a release series' subrepository in the file:

```
[mysql57-community]
name=MySQL 5.7 Community Server
baseurl=http://repo.mysql.com/yum/mysql-5.7-community/el/6/$basearch/
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-mysql-2022
       file:///etc/pki/rpm-gpg/RPM-GPG-KEY-mysql
```

Find the entry for the subrepository you want to configure, and edit the `enabled` option. Specify `enabled=0` to disable a subrepository, or `enabled=1` to enable a subrepository. For example, to install MySQL 8.0, make sure you have `enabled=0` for the above subrepository entry for MySQL 5.7, and have `enabled=1` for the entry for the 8.0 series:

```
# Enable to use MySQL 8.0
[mysql80-community]
name=MySQL 8.0 Community Server
baseurl=http://repo.mysql.com/yum/mysql-8.0-community/el/6/$basearch/
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-mysql-2022
       file:///etc/pki/rpm-gpg/RPM-GPG-KEY-mysql
```

You should only enable subrepository for one release series at any time. When subrepositories for more than one release series are enabled, Yum uses the latest series.

Verify that the correct subrepositories have been enabled and disabled by running the following command and checking its output (for dnf-enabled systems, replace `yum` in the command with `dnf`):

```
$> yum repolist enabled | grep mysql
```

3.8 Disabling the Default MySQL Module

(EL8 systems only) EL8-based systems such as RHEL8 and Oracle Linux 8 include a MySQL module that is enabled by default. Unless this module is disabled, it masks packages provided by MySQL repositories. To disable the included module and make the MySQL repository packages visible, use the following command (for dnf-enabled systems, replace `yum` in the command with `dnf`):

```
$> sudo yum module disable mysql
```

Installing MySQL

Install MySQL by the following command (for dnf-enabled systems, replace `yum` in the command with `dnf`):

```
$> sudo yum install mysql-community-server
```

This installs the package for MySQL server (`mysql-community-server`) and also packages for the components required to run the server, including packages for the client (`mysql-community-client`), the common error messages and character sets for client and server (`mysql-community-common`), and the shared client libraries (`mysql-community-libs`).

Starting the MySQL Server

Start the MySQL server with the following command:

```
$> systemctl start mysqld
```

You can check the status of the MySQL server with the following command:

```
$> systemctl status mysqld
```

If the operating system is systemd enabled, standard `systemctl` (or alternatively, `service` with the arguments reversed) commands such as `stop`, `start`, `status`, and `restart` should be used to manage the MySQL server service. The `mysqld` service is enabled by default, and it starts at system reboot. See Section 2.5.9, “Managing MySQL Server with systemd” for additional information.

At the initial start up of the server, the following happens, given that the data directory of the server is empty:

- The server is initialized.
- SSL certificate and key files are generated in the data directory.
- `validate_password` is installed and enabled.
- A superuser account `'root'@'localhost'` is created. A password for the superuser is set and stored in the error log file. To reveal it, use the following command:

  ```
  $> sudo grep 'temporary password' /var/log/mysqld.log
  ```

  Change the root password as soon as possible by logging in with the generated, temporary password and set a custom password for the superuser account:

  ```
  $> mysql -uroot -p
  ```

  ```
  mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'MyNewPass4!';
  ```

Note

`validate_password` is installed by default. The default password policy implemented by `validate_password` requires that passwords contain at least one uppercase letter, one lowercase letter, one digit, and one special character, and that the total password length is at least 8 characters.

For more information on the postinstallation procedures, see Section 2.9, “Postinstallation Setup and Testing”.

Note

Compatibility Information for EL7-based platforms: The following RPM packages from the native software repositories of the platforms are incompatible with the package from the MySQL Yum repository that installs the MySQL server. Once

you have installed MySQL using the MySQL Yum repository, you cannot install these packages (and vice versa).

- akonadi-mysql

Installing Additional MySQL Products and Components with Yum

You can use Yum to install and manage individual components of MySQL. Some of these components are hosted in sub-repositories of the MySQL Yum repository: for example, the MySQL Connectors are to be found in the MySQL Connectors Community sub-repository, and the MySQL Workbench in MySQL Tools Community. You can use the following command to list the packages for all the MySQL components available for your platform from the MySQL Yum repository (for dnf-enabled systems, replace `yum` in the command with `dnf`):

```
$> sudo yum --disablerepo=\* --enablerepo='mysql*-community*' list available
```

Install any packages of your choice with the following command, replacing *`package-name`* with name of the package (for dnf-enabled systems, replace `yum` in the command with `dnf`):

```
$> sudo yum install package-name
```

For example, to install MySQL Workbench on Fedora:

```
$> sudo dnf install mysql-workbench-community
```

To install the shared client libraries (for dnf-enabled systems, replace `yum` in the command with `dnf`):

```
$> sudo yum install mysql-community-libs
```

Platform Specific Notes

ARM Support

ARM 64-bit (aarch64) is supported on Oracle Linux 7 and requires the Oracle Linux 7 Software Collections Repository (ol7_software_collections). For example, to install the server:

```
$> yum-config-manager --enable ol7_software_collections
$> yum install mysql-community-server
```

Note

ARM 64-bit (aarch64) is supported on Oracle Linux 7 as of MySQL 8.0.12.

Known Limitation

The 8.0.12 release requires you to adjust the *libstdc++7* path by executing `ln -s /opt/oracle/oracle-armtoolset-1/root/usr/lib64 /usr/lib64/gcc7` after executing the `yum install` step.

Updating MySQL with Yum

Besides installation, you can also perform updates for MySQL products and components using the MySQL Yum repository. See Section 2.10.7, “Upgrading MySQL with the MySQL Yum Repository” for details.

2.5.2 Installing MySQL on Linux Using the MySQL APT Repository

The MySQL APT repository provides `deb` packages for installing and managing the MySQL server, client, and other components on the current Debian and Ubuntu releases.

Instructions for using the MySQL APT Repository are available in A Quick Guide to Using the MySQL APT Repository.

2.5.3 Installing MySQL on Linux Using the MySQL SLES Repository

The MySQL SLES repository provides RPM packages for installing and managing the MySQL server, client, and other components on SUSE Enterprise Linux Server.

Instructions for using the MySQL SLES repository are available in A Quick Guide to Using the MySQL SLES Repository.

2.5.4 Installing MySQL on Linux Using RPM Packages from Oracle

The recommended way to install MySQL on RPM-based Linux distributions is by using the RPM packages provided by Oracle. There are two sources for obtaining them, for the Community Edition of MySQL:

- From the MySQL software repositories:
  - The MySQL Yum repository (see Section 2.5.1, “Installing MySQL on Linux Using the MySQL Yum Repository” for details).
  - The MySQL SLES repository (see Section 2.5.3, “Installing MySQL on Linux Using the MySQL SLES Repository” for details).
- From the Download MySQL Community Server page in the MySQL Developer Zone.

Note

RPM distributions of MySQL are also provided by other vendors. Be aware that they may differ from those built by Oracle in features, capabilities, and conventions (including communication setup), and that the installation instructions in this manual do not necessarily apply to them. The vendor's instructions should be consulted instead.

MySQL RPM Packages

Table 2.9 RPM Packages for MySQL Community Edition

Package Name	Summary
mysql-community-client	MySQL client applications and tools
mysql-community-client-plugins	Shared plugins for MySQL client applications
mysql-community-common	Common files for server and client libraries
mysql-community-devel	Development header files and libraries for MySQL database client applications
mysql-community-embedded-compat	MySQL server as an embedded library with compatibility for applications using version 18 of the library
mysql-community-icu-data-files	MySQL packaging of ICU data files needed by MySQL regular expressions
mysql-community-libs	Shared libraries for MySQL database client applications
mysql-community-libs-compat	Shared compatibility libraries for previous MySQL installations
mysql-community-server	Database server and related tools
mysql-community-server-debug	Debug server and plugin binaries
mysql-community-test	Test suite for the MySQL server
mysql-community	The source code RPM looks similar to mysql-community-8.0.32-1.el7.src.rpm, depending on selected OS

Package Name	Summary
Additional *debuginfo* RPMs	There are several `debuginfo` packages: mysql-community-client-debuginfo, mysql-community-libs-debuginfo mysql-community-server-debug-debuginfo mysql-community-server-debuginfo, and mysql-community-test-debuginfo.

Table 2.10 RPM Packages for the MySQL Enterprise Edition

Package Name	Summary
`mysql-commercial-backup`	MySQL Enterprise Backup (added in 8.0.11)
`mysql-commercial-client`	MySQL client applications and tools
`mysql-commercial-client-plugins`	Shared plugins for MySQL client applications
`mysql-commercial-common`	Common files for server and client libraries
`mysql-commercial-devel`	Development header files and libraries for MySQL database client applications
`mysql-commercial-embedded-compat`	MySQL server as an embedded library with compatibility for applications using version 18 of the library
`mysql-commercial-icu-data-files`	MySQL packaging of ICU data files needed by MySQL regular expressions
`mysql-commercial-libs`	Shared libraries for MySQL database client applications
`mysql-commercial-libs-compat`	Shared compatibility libraries for previous MySQL installations; the version of the libraries matches the version of the libraries installed by default by the distribution you are using
`mysql-commercial-server`	Database server and related tools
`mysql-commercial-test`	Test suite for the MySQL server
Additional *debuginfo* RPMs	There are several `debuginfo` packages: mysql-commercial-client-debuginfo, mysql-commercial-libs-debuginfo mysql-commercial-server-debug-debuginfo mysql-commercial-server-debuginfo, and mysql-commercial-test-debuginfo.

The full names for the RPMs have the following syntax:

```
packagename-version-distribution-arch.rpm
```

The *`distribution`* and *`arch`* values indicate the Linux distribution and the processor type for which the package was built. See the table below for lists of the distribution identifiers:

Table 2.11 MySQL Linux RPM Package Distribution Identifiers

Distribution Value	Intended Use
el*`{version}`* where *`{version}`* is the major Enterprise Linux version, such as `el8`	EL6, EL7, EL8, and EL9-based platforms (for example, the corresponding versions of Oracle Linux, Red Hat Enterprise Linux, and CentOS)
fc*`{version}`* where *`{version}`* is the major Fedora version, such as `fc37`	Fedora 36 and 37
`sles12`	SUSE Linux Enterprise Server 12

To see all files in an RPM package (for example, `mysql-community-server`), use the following command:

```
$> rpm -qpl mysql-community-server-version-distribution-arch.rpm
```

The discussion in the rest of this section applies only to an installation process using the RPM packages directly downloaded from Oracle, instead of through a MySQL repository.

Dependency relationships exist among some of the packages. If you plan to install many of the packages, you may wish to download the RPM bundle `tar` file instead, which contains all the RPM packages listed above, so that you need not download them separately.

In most cases, you need to install the `mysql-community-server`, `mysql-community-client`, `mysql-community-client-plugins`, `mysql-community-libs`, `mysql-community-icu-data-files`, `mysql-community-common`, and `mysql-community-libs-compat` packages to get a functional, standard MySQL installation. To perform such a standard, basic installation, go to the folder that contains all those packages (and, preferably, no other RPM packages with similar names), and issue the following command:

```
$> sudo yum install mysql-community-{server,client,client-plugins,icu-data-files,common,libs}-*
```

Replace `yum` with `zypper` for SLES, and with `dnf` for Fedora.

While it is much preferable to use a high-level package management tool like `yum` to install the packages, users who prefer direct `rpm` commands can replace the `yum install` command with the `rpm -Uvh` command; however, using `rpm -Uvh` instead makes the installation process more prone to failure, due to potential dependency issues the installation process might run into.

To install only the client programs, you can skip `mysql-community-server` in your list of packages to install; issue the following command:

```
$> sudo yum install mysql-community-{client,client-plugins,common,libs}-*
```

Replace `yum` with `zypper` for SLES, and with `dnf` for Fedora.

A standard installation of MySQL using the RPM packages result in files and resources created under the system directories, shown in the following table.

Table 2.12 MySQL Installation Layout for Linux RPM Packages from the MySQL Developer Zone

Files or Resources	Location
Client programs and scripts	`/usr/bin`
`mysqld` server	`/usr/sbin`
Configuration file	`/etc/my.cnf`
Data directory	`/var/lib/mysql`
Error log file	For RHEL, Oracle Linux, CentOS or Fedora platforms: `/var/log/mysqld.log` For SLES: `/var/log/mysql/mysqld.log`
Value of `secure_file_priv`	`/var/lib/mysql-files`
System V init script	For RHEL, Oracle Linux, CentOS or Fedora platforms: `/etc/init.d/mysqld` For SLES: `/etc/init.d/mysql`
Systemd service	For RHEL, Oracle Linux, CentOS or Fedora platforms: `mysqld` For SLES: `mysql`
Pid file	`/var/run/mysql/mysqld.pid`
Socket	`/var/lib/mysql/mysql.sock`
Keyring directory	`/var/lib/mysql-keyring`

Files or Resources	Location
Unix manual pages	`/usr/share/man`
Include (header) files	`/usr/include/mysql`
Libraries	`/usr/lib/mysql`
Miscellaneous support files (for example, error messages, and character set files)	`/usr/share/mysql`

The installation also creates a user named `mysql` and a group named `mysql` on the system.

Note

Installation of previous versions of MySQL using older packages might have created a configuration file named `/usr/my.cnf`. It is highly recommended that you examine the contents of the file and migrate the desired settings inside to the file `/etc/my.cnf` file, then remove `/usr/my.cnf`.

MySQL is NOT automatically started at the end of the installation process. For Red Hat Enterprise Linux, Oracle Linux, CentOS, and Fedora systems, use the following command to start MySQL:

```
$> systemctl start mysqld
```

For SLES systems, the command is the same, but the service name is different:

```
$> systemctl start mysql
```

If the operating system is systemd enabled, standard `systemctl` (or alternatively, `service` with the arguments reversed) commands such as `stop`, `start`, `status`, and `restart` should be used to manage the MySQL server service. The `mysqld` service is enabled by default, and it starts at system reboot. Notice that certain things might work differently on systemd platforms: for example, changing the location of the data directory might cause issues. See Section 2.5.9, “Managing MySQL Server with systemd” for additional information.

During an upgrade installation using RPM and DEB packages, if the MySQL server is running when the upgrade occurs then the MySQL server is stopped, the upgrade occurs, and the MySQL server is restarted. One exception: if the edition also changes during an upgrade (such as community to commercial, or vice-versa), then MySQL server is not restarted.

At the initial start up of the server, the following happens, given that the data directory of the server is empty:

- The server is initialized.
- An SSL certificate and key files are generated in the data directory.
- `validate_password` is installed and enabled.
- A superuser account `'root'@'localhost'` is created. A password for the superuser is set and stored in the error log file. To reveal it, use the following command for RHEL, Oracle Linux, CentOS, and Fedora systems:

  ```
  $> sudo grep 'temporary password' /var/log/mysqld.log
  ```

  Use the following command for SLES systems:

  ```
  $> sudo grep 'temporary password' /var/log/mysql/mysqld.log
  ```

  The next step is to log in with the generated, temporary password and set a custom password for the superuser account:

```
$> mysql -uroot -p
```

```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'MyNewPass4!';
```

Note

`validate_password` is installed by default. The default password policy implemented by `validate_password` requires that passwords contain at least one uppercase letter, one lowercase letter, one digit, and one special character, and that the total password length is at least 8 characters.

If something goes wrong during installation, you might find debug information in the error log file `/var/log/mysqld.log`.

For some Linux distributions, it might be necessary to increase the limit on number of file descriptors available to `mysqld`. See Section B.3.2.16, “File Not Found and Similar Errors”

Installing Client Libraries from Multiple MySQL Versions. It is possible to install multiple client library versions, such as for the case that you want to maintain compatibility with older applications linked against previous libraries. To install an older client library, use the `--oldpackage` option with `rpm`. For example, to install `mysql-community-libs-5.5` on an EL6 system that has `libmysqlclient.21` from MySQL 8.0, use a command like this:

```
$> rpm --oldpackage -ivh mysql-community-libs-5.5.50-2.el6.x86_64.rpm
```

Debug Package. A special variant of MySQL Server compiled with the debug package has been included in the server RPM packages. It performs debugging and memory allocation checks and produces a trace file when the server is running. To use that debug version, start MySQL with `/usr/sbin/mysqld-debug`, instead of starting it as a service or with `/usr/sbin/mysqld`. See Section 5.9.4, “The DBUG Package” for the debug options you can use.

Note

The default plugin directory for debug builds changed from `/usr/lib64/mysql/plugin` to `/usr/lib64/mysql/plugin/debug` in MySQL 8.0.4. Previously, it was necessary to change `plugin_dir` to `/usr/lib64/mysql/plugin/debug` for debug builds.

Rebuilding RPMs from source SRPMs. Source code SRPM packages for MySQL are available for download. They can be used as-is to rebuild the MySQL RPMs with the standard `rpmbuild` tool chain.

2.5.5 Installing MySQL on Linux Using Debian Packages from Oracle

Oracle provides Debian packages for installing MySQL on Debian or Debian-like Linux systems. The packages are available through two different channels:

- The MySQL APT Repository. This is the preferred method for installing MySQL on Debian-like systems, as it provides a simple and convenient way to install and update MySQL products. For details, see Section 2.5.2, “Installing MySQL on Linux Using the MySQL APT Repository”.
- The MySQL Developer Zone's Download Area. For details, see Section 2.1.3, “How to Get MySQL”. The following are some information on the Debian packages available there and the instructions for installing them:
  - Various Debian packages are provided in the MySQL Developer Zone for installing different components of MySQL on the current Debian and Ubuntu platforms. The preferred method is to use the tarball bundle, which contains the packages needed for a basic setup of MySQL. The tarball bundles have names in the format of `mysql-server_`*`MVER`*`-`*`DVER`*`_CPU.deb-bundle.tar`. *`MVER`* is the MySQL version and *`DVER`* is the Linux distribution version. The *`CPU`* value indicates the processor type or family for which the package is built, as shown in the following table:

    Table 2.13 MySQL Debian and Ubuntu Installation Packages CPU Identifiers

`CPU` Value	Intended Processor Type or Family
`i386`	Pentium processor or better, 32 bit

CPU Value	Intended Processor Type or Family
amd64	64-bit x86 processor

- After downloading the tarball, unpack it with the following command:

```
$> tar -xvf mysql-server_MVER-DVER_CPU.deb-bundle.tar
```

- You may need to install the libaio library if it is not already present on your system:

```
$> sudo apt-get install libaio1
```

- Preconfigure the MySQL server package with the following command:

```
$> sudo dpkg-preconfigure mysql-community-server_*.deb
```

  You are asked to provide a password for the root user for your MySQL installation. You might also be asked other questions regarding the installation.

Important

Make sure you remember the root password you set. Users who want to set a password later can leave the **password** field blank in the dialogue box and just press **OK**; in that case, root access to the server is authenticated using the MySQL Socket Peer-Credential Authentication Plugin for connections using a Unix socket file. You can set the root password later using mysql_secure_installation.

- For a basic installation of the MySQL server, install the database common files package, the client package, the client metapackage, the server package, and the server metapackage (in that order); you can do that with a single command:

```
$> sudo dpkg -i mysql-{common,community-client-plugins,community-client-core,community-client,clien
```

  There are also packages with server-core and client-core in the package names. These contain binaries only and are installed automatically by the standard packages. Installing them by themselves does not result in a functioning MySQL setup.

  If you are being warned of unmet dependencies by dpkg (such as libmecab2), you can fix them using apt-get:

```
sudo apt-get -f install
```

  Here are where the files are installed on the system:

  - All configuration files (like my.cnf) are under /etc/mysql
  - All binaries, libraries, headers, etc., are under /usr/bin and /usr/sbin
  - The data directory is under /var/lib/mysql

Note

Debian distributions of MySQL are also provided by other vendors. Be aware that they may differ from those built by Oracle in features, capabilities, and conventions (including communication setup), and that the instructions in this manual do not necessarily apply to installing them. The vendor's instructions should be consulted instead.

2.5.6 Deploying MySQL on Linux with Docker

The Docker deployment framework supports easy installation and configuration of MySQL Server. This section explains how to use a MySQL Server Docker image.

You need to have Docker installed on your system before you can use a MySQL Server Docker image. See Install Docker for instructions.

Warning

Beware of the security concerns with running Docker containers. See Docker security for details.

2.5.6.1 Basic Steps for MySQL Server Deployment with Docker

Warning

The MySQL Docker images maintained by the MySQL team are built specifically for Linux platforms. Other platforms are not supported, and users using these MySQL Docker images on them are doing so at their own risk. See the discussion here for some known limitations for running these containers on non-Linux operating systems.

- Downloading a MySQL Server Docker Image
- Starting a MySQL Server Instance
- Connecting to MySQL Server from within the Container
- Container Shell Access
- Stopping and Deleting a MySQL Container
- Upgrading a MySQL Server Container
- More Topics on Deploying MySQL Server with Docker

Downloading a MySQL Server Docker Image

Important

For users of MySQL Enterprise Edition: A subscription is required to use the Docker images for MySQL Enterprise Edition. Subscriptions work by a Bring Your Own License model; see How to Buy MySQL Products and Services for details.

Downloading the server image in a separate step is not strictly necessary; however, performing this step before you create your Docker container ensures your local image is up to date. To download the MySQL Community Edition image, run this command:

```
docker pull mysql/mysql-server:tag
```

The *`tag`* is the label for the image version you want to pull (for example, `5.6`, `5.7`, `8.0`, or `latest`). If **`:tag`** is omitted, the `latest` label is used, and the image for the latest GA version of MySQL Community Server is downloaded. Refer to the list of tags for available versions on the mysql/mysql-server page in the Docker Hub.

To download the MySQL Community Edition image from the Oracle Container Registry (OCR), run this command:

```
docker pull container-registry.oracle.com/mysql/community-server:tag
```

To download the MySQL Enterprise Edition image from the OCR, you need to first accept the license agreement on the OCR and log in to the container repository with your Docker client:

- Visit the OCR at https://container-registry.oracle.com/ and choose **MySQL**.
- Under the list of MySQL repositories, choose `enterprise-server`.

- If you have not signed in to the OCR yet, click the **Sign in** button on the right of the page, and then enter your Oracle account credentials when prompted to.
- Follow the instructions on the right of the page to accept the license agreement.
- Log in to the OCR with your Docker client (the `docker` command) using the `docker login` command:

```
# docker login container-registry.oracle.com
Username: Oracle-Account-ID
Password: password
Login successful.
```

Download the Docker image for MySQL Enterprise Edition from the OCR with this command:

```
docker pull  container-registry.oracle.com/mysql/enterprise-server:tag
```

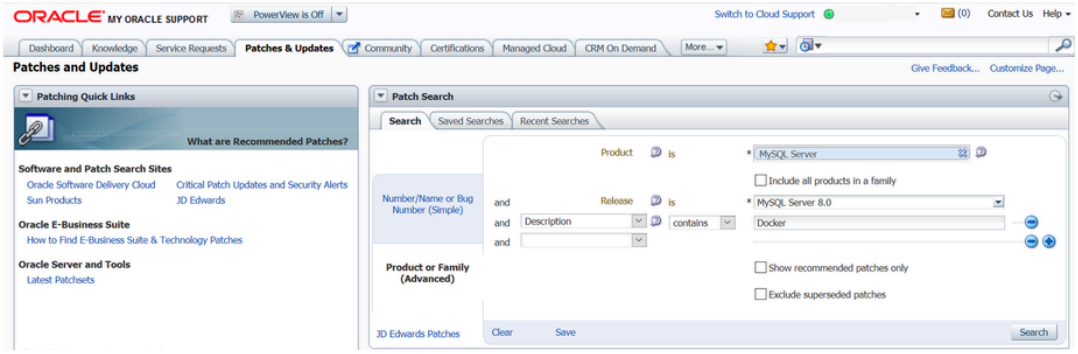
There are different choices for *`tag`*, corresponding to different versions of MySQL Docker images provided by the OCR:

- `8.0`, `8.0.x` (`x` is the latest version number in the 8.0 series), `latest`: MySQL 8.0, the latest GA
- `5.7`, `5.7.y` (`y` is the latest version number in the 5.7 series): MySQL 5.7

To download the MySQL Enterprise Edition image from My Oracle Support website, go onto the website, sign in to your Oracle account, and perform these steps once you are on the landing page:

- Select the **Patches and Updates** tab.
- Go to the **Patch Search** region and, on the **Search** tab, switch to the **Product or Family (Advanced)** subtab.
- Enter “MySQL Server” for the **Product** field, and the desired version number in the **Release** field.
- Use the dropdowns for additional filters to select **Description**—**contains**, and enter “Docker” in the text field.

  The following figure shows the search settings for the MySQL Enterprise Edition image for MySQL Server 8.0:



- Click the **Search** button and, from the result list, select the version you want, and click the **Download** button.
- In the **File Download** dialogue box that appears, click and download the `.zip` file for the Docker image.

Unzip the downloaded `.zip` archive to obtain the tarball inside (`mysql-enterprise-server-version.tar`), and then load the image by running this command:

```
docker load -i mysql-enterprise-server-version.tar
```

You can list downloaded Docker images with this command:

```
$> docker images
REPOSITORY           TAG                 IMAGE ID            CREATED             SIZE
mysql/mysql-server   latest              3157d7f55f8d        4 weeks ago         241MB
```

Starting a MySQL Server Instance

To start a new Docker container for a MySQL Server, use the following command:

```
docker run --name=container_name  --restart on-failure -d image_name:tag
```

The image name can be obtained using the `docker images` command, as explained in Downloading a MySQL Server Docker Image.

The `--name` option, for supplying a custom name for your server container, is optional; if no container name is supplied, a random one is generated.

The `--restart` option is for configuring the restart policy for your container; it should be set to the value `on-failure`, to enable support for server restart within a client session (which happens, for example, when the RESTART statement is executed by a client or during the configuration of an InnoDB cluster instance). With the support for restart enabled, issuing a restart within a client session causes the server and the container to stop and then restart. *Support for server restart is available for MySQL 8.0.21 and later.*

For example, to start a new Docker container for the MySQL Community Server, use this command:

```
docker run --name=mysql1 --restart on-failure -d mysql/mysql-server:8.0
```

To start a new Docker container for the MySQL Enterprise Server with a Docker image downloaded from the OCR, use this command:

```
docker run --name=mysql1 --restart on-failure -d container-registry.oracle.com/mysql/enterprise-server:8.0
```

To start a new Docker container for the MySQL Enterprise Server with a Docker image downloaded from My Oracle Support, use this command:

```
docker run --name=mysql1 --restart on-failure -d mysql/enterprise-server:8.0
```

If the Docker image of the specified name and tag has not been downloaded by an earlier `docker pull` or `docker run` command, the image is now downloaded. Initialization for the container begins, and the container appears in the list of running containers when you run the `docker ps` command. For example:

```
$> docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED             STATUS
a24888f0d6f4   mysql/mysql-server   "/entrypoint.sh my..."   14 seconds ago      Up 13 seconds (health: sta
```

The container initialization might take some time. When the server is ready for use, the `STATUS` of the container in the output of the `docker ps` command changes from `(health: starting)` to `(healthy)`.

The `-d` option used in the `docker run` command above makes the container run in the background. Use this command to monitor the output from the container:

```
docker logs mysql1
```

Once initialization is finished, the command's output is going to contain the random password generated for the root user; check the password with, for example, this command:

```
$> docker logs mysql1 2>&1 | grep GENERATED
GENERATED ROOT PASSWORD: Axegh3kAJyDLaRuBemecis&EShOs
```

Connecting to MySQL Server from within the Container

Once the server is ready, you can run the `mysql` client within the MySQL Server container you just started, and connect it to the MySQL Server. Use the `docker exec -it` command to start a `mysql` client inside the Docker container you have started, like the following:

```
docker exec -it mysql1 mysql -uroot -p
```

When asked, enter the generated root password (see the last step in Starting a MySQL Server Instance above on how to find the password). Because the `MYSQL_ONETIME_PASSWORD` option is true by default, after you have connected a `mysql` client to the server, you must reset the server root password by issuing this statement:

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

Substitute *`password`* with the password of your choice. Once the password is reset, the server is ready for use.

Container Shell Access

To have shell access to your MySQL Server container, use the `docker exec -it` command to start a bash shell inside the container:

```
$> docker exec -it mysql1 bash
bash-4.2#
```

You can then run Linux commands inside the container. For example, to view contents in the server's data directory inside the container, use this command:

```
bash-4.2# ls /var/lib/mysql
auto.cnf    ca.pem      client-key.pem  ib_logfile0  ibdata1  mysql       mysql.sock.lock    private_ke
ca-key.pem  client-cert.pem  ib_buffer_pool  ib_logfile1  ibtmp1   mysql.sock  performance_schema  publ
```

Stopping and Deleting a MySQL Container

To stop the MySQL Server container we have created, use this command:

```
docker stop mysql1
```

`docker stop` sends a SIGTERM signal to the `mysqld` process, so that the server is shut down gracefully.

Also notice that when the main process of a container (`mysqld` in the case of a MySQL Server container) is stopped, the Docker container stops automatically.

To start the MySQL Server container again:

```
docker start mysql1
```

To stop and start again the MySQL Server container with a single command:

```
docker restart mysql1
```

To delete the MySQL container, stop it first, and then use the `docker rm` command:

```
docker stop mysql1
```

```
docker rm mysql1
```

If you want the Docker volume for the server's data directory to be deleted at the same time, add the `-v` option to the `docker rm` command.

Upgrading a MySQL Server Container

Important

- Before performing any upgrade to MySQL, follow carefully the instructions in Section 2.10, “Upgrading MySQL”. Among other instructions discussed there, it is especially important to back up your database before the upgrade.
- The instructions in this section require that the server's data and configuration have been persisted on the host. See Persisting Data and Configuration Changes for details.

Follow these steps to upgrade a Docker installation of MySQL 5.7 to 8.0:

- Stop the MySQL 5.7 server (container name is `mysql57` in this example):

  ```
  docker stop mysql57
  ```

- Download the MySQL 8.0 Server Docker image. See instructions in Downloading a MySQL Server Docker Image. Make sure you use the right tag for MySQL 8.0.

- Start a new MySQL 8.0 Docker container (named `mysql80` in this example) with the old server data and configuration (with proper modifications if needed—see Section 2.10, "Upgrading MySQL") that have been persisted on the host (by bind-mounting in this example). For the MySQL Community Server, run this command:

  ```
  docker run --name=mysql80 \
     --mount type=bind,src=/path-on-host-machine/my.cnf,dst=/etc/my.cnf \
     --mount type=bind,src=/path-on-host-machine/datadir,dst=/var/lib/mysql \
     -d mysql/mysql-server:8.0
  ```

  If needed, adjust `mysql/mysql-server` to the correct image name—for example, replace it with `container-registry.oracle.com/mysql/enterprise-server` for MySQL Enterprise Edition images downloaded from the OCR, or `mysql/enterprise-server` for MySQL Enterprise Edition images downloaded from My Oracle Support.

- Wait for the server to finish startup. You can check the status of the server using the `docker ps` command (see Starting a MySQL Server Instance for how to do that).

Follow the same steps for upgrading within the 8.0 series (that is, from release 8.0.*x* to 8.0.*y*): stop the original container, and start a new one with a newer image on the old server data and configuration. If you used the `8.0` or the `latest` tag when starting your original container and there is now a new MySQL 8.0 release you want to upgrade to it, you must first pull the image for the new release with the command:

```
docker pull mysql/mysql-server:8.0
```

You can then upgrade by starting a *new* container with the same tag on the old data and configuration (adjust the repository name if you are using the MySQL Enterprise Edition; see Downloading a MySQL Server Docker Image):

```
docker run --name=mysql80new \
   --mount type=bind,src=/path-on-host-machine/my.cnf,dst=/etc/my.cnf \
   --mount type=bind,src=/path-on-host-machine/datadir,dst=/var/lib/mysql \
   -d mysql/mysql-server:8.0
```

Note

For MySQL 8.0.15 and earlier: You need to complete the upgrade process by running the mysql_upgrade utility in the MySQL 8.0 Server container (the step is *not* required for MySQL 8.0.16 and later):

- ```
  docker exec -it mysql80 mysql_upgrade -uroot -p
  ```

  When prompted, enter the root password for your old server.

- Finish the upgrade by restarting the new container:

  ```
  docker restart mysql80
  ```

More Topics on Deploying MySQL Server with Docker

For more topics on deploying MySQL Server with Docker like server configuration, persisting data and configuration, server error log, and container environment variables, see Section 2.5.6.2, "More Topics on Deploying MySQL Server with Docker".
  ```

2.5.6.2 More Topics on Deploying MySQL Server with Docker

Note

Most of the following sample commands have `mysql/mysql-server` as the Docker image repository when that has to be specified (like with the `docker pull` and `docker run` commands); change that if your image is from another repository—for example, replace it with `container-registry.oracle.com/mysql/enterprise-server` for MySQL Enterprise Edition images downloaded from the Oracle Container Registry (OCR), or `mysql/enterprise-server` for MySQL Enterprise Edition images downloaded from My Oracle Support.

- The Optimized MySQL Installation for Docker
- Configuring the MySQL Server
- Persisting Data and Configuration Changes
- Running Additional Initialization Scripts
- Connect to MySQL from an Application in Another Docker Container
- Server Error Log
- Using MySQL Enterprise Backup with Docker
- Using mysqldump with Docker
- Known Issues
- Docker Environment Variables

The Optimized MySQL Installation for Docker

Docker images for MySQL are optimized for code size, which means they only include crucial components that are expected to be relevant for the majority of users who run MySQL instances in Docker containers. A MySQL Docker installation is different from a common, non-Docker installation in the following aspects:

- Included binaries are limited to:
  - `/usr/bin/my_print_defaults`
  - `/usr/bin/mysql`
  - `/usr/bin/mysql_config`
  - `/usr/bin/mysql_install_db`
  - `/usr/bin/mysql_tzinfo_to_sql`
  - `/usr/bin/mysql_upgrade`
  - `/usr/bin/mysqladmin`
  - `/usr/bin/mysqlcheck`
  - `/usr/bin/mysqldump`
  - `/usr/bin/mysqlpump`
  - `/usr/bin/mysqlbackup` (for MySQL Enterprise Edition 8.0 only)

  - `/usr/sbin/mysqld`

- All binaries are stripped; they contain no debug information.

Warning

Any software updates or installations users perform to the Docker container (including those for MySQL components) may conflict with the optimized MySQL installation created by the Docker image. Oracle does not provide support for MySQL products running in such an altered container, or a container created from an altered Docker image.

Configuring the MySQL Server

When you start the MySQL Docker container, you can pass configuration options to the server through the `docker run` command. For example:

```
docker run --name mysql1 -d mysql/mysql-server:tag --character-set-server=utf8mb4 --collation-server=utf8mb
```

The command starts the MySQL Server with `utf8mb4` as the default character set and `utf8mb4_col` as the default collation for databases.

Another way to configure the MySQL Server is to prepare a configuration file and mount it at the location of the server configuration file inside the container. See Persisting Data and Configuration Changes for details.

Persisting Data and Configuration Changes

Docker containers are in principle ephemeral, and any data or configuration are expected to be lost if the container is deleted or corrupted (see discussions here). Docker volumes provides a mechanism to persist data created inside a Docker container. At its initialization, the MySQL Server container creates a Docker volume for the server data directory. The JSON output from the `docker inspect` command on the container includes a `Mount` key, whose value provides information on the data directory volume:

```
$> docker inspect mysql1
...
 "Mounts": [
            {
                "Type": "volume",
                "Name": "4f2d463cfc4bdd4baebcb098c97d7da3337195ed2c6572bc0b89f7e845d27652",
                "Source": "/var/lib/docker/volumes/4f2d463cfc4bdd4baebcb098c97d7da3337195ed2c6572bc0b89f7e8
                "Destination": "/var/lib/mysql",
                "Driver": "local",
                "Mode": "",
                "RW": true,
                "Propagation": ""
            }
        ],
...
```

The output shows that the source directory `/var/lib/docker/volumes/4f2d463cfc4bdd4baebcb098c97d7da3337195ed2c6572bc0b89f7e845d27652/_data`, in which data is persisted on the host, has been mounted at `/var/lib/mysql`, the server data directory inside the container.

Another way to preserve data is to bind-mount a host directory using the `--mount` option when creating the container. The same technique can be used to persist the configuration of the server. The following command creates a MySQL Server container and bind-mounts both the data directory and the server configuration file:

```
docker run --name=mysql1 \
--mount type=bind,src=/path-on-host-machine/my.cnf,dst=/etc/my.cnf \
--mount type=bind,src=/path-on-host-machine/datadir,dst=/var/lib/mysql \
```

```
-d mysql/mysql-server:tag
```

The command mounts *`path-on-host-machine/my.cnf`* at `/etc/my.cnf` (the server configuration file inside the container), and *`path-on-host-machine/datadir`* at `/var/lib/mysql` (the data directory inside the container). The following conditions must be met for the bind-mounting to work:

- The configuration file *`path-on-host-machine/my.cnf`* must already exist, and it must contain the specification for starting the server by the user `mysql`:

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

  You can also include other server configuration options in the file.

- The data directory *`path-on-host-machine/datadir`* must already exist. For server initialization to happen, the directory must be empty. You can also mount a directory prepopulated with data and start the server with it; however, you must make sure you start the Docker container with the same configuration as the server that created the data, and any host files or directories required are mounted when starting the container.

Running Additional Initialization Scripts

If there are any `.sh` or `.sql` scripts you want to run on the database immediately after it has been created, you can put them into a host directory and then mount the directory at `/docker-entrypoint-initdb.d/` inside the container. For example:

```
docker run --name=mysql1 \
--mount type=bind,src=/path-on-host-machine/scripts/,dst=/docker-entrypoint-initdb.d/ \
-d mysql/mysql-server:tag
```

Connect to MySQL from an Application in Another Docker Container

By setting up a Docker network, you can allow multiple Docker containers to communicate with each other, so that a client application in another Docker container can access the MySQL Server in the server container. First, create a Docker network:

```
docker network create my-custom-net
```

Then, when you are creating and starting the server and the client containers, use the `--network` option to put them on network you created. For example:

```
docker run --name=mysql1 --network=my-custom-net -d mysql/mysql-server
```

```
docker run --name=myapp1 --network=my-custom-net -d myapp
```

The `myapp1` container can then connect to the `mysql1` container with the `mysql1` hostname and vice versa, as Docker automatically sets up a DNS for the given container names. In the following example, we run the `mysql` client from inside the `myapp1` container to connect to host `mysql1` in its own container:

```
docker exec -it myapp1 mysql --host=mysql1 --user=myuser --password
```

For other networking techniques for containers, see the Docker container networking section in the Docker Documentation.

Server Error Log

When the MySQL Server is first started with your server container, a server error log is NOT generated if either of the following conditions is true:

- A server configuration file from the host has been mounted, but the file does not contain the system variable `log_error` (see Persisting Data and Configuration Changes on bind-mounting a server configuration file).

- A server configuration file from the host has not been mounted, but the Docker environment variable `MYSQL_LOG_CONSOLE` is `true` (which is the variable's default state for MySQL 8.0 server containers). The MySQL Server's error log is then redirected to `stderr`, so that the error log goes into the Docker container's log and is viewable using the `docker logs mysqld-container` command.

To make MySQL Server generate an error log when either of the two conditions is true, use the `--log-error` option to configure the server to generate the error log at a specific location inside the container. To persist the error log, mount a host file at the location of the error log inside the container as explained in Persisting Data and Configuration Changes. However, you must make sure your MySQL Server inside its container has write access to the mounted host file.

Using MySQL Enterprise Backup with Docker

MySQL Enterprise Backup is a commercially-licensed backup utility for MySQL Server, available with MySQL Enterprise Edition. MySQL Enterprise Backup is included in the Docker installation of MySQL Enterprise Edition.

In the following example, we assume that you already have a MySQL Server running in a Docker container (see Section 2.5.6.1, “Basic Steps for MySQL Server Deployment with Docker” on how to start a MySQL Server instance with Docker). For MySQL Enterprise Backup to back up the MySQL Server, it must have access to the server's data directory. This can be achieved by, for example, bind-mounting a host directory on the data directory of the MySQL Server when you start the server:

```
docker run --name=mysqlserver \
--mount type=bind,src=/path-on-host-machine/datadir/,dst=/var/lib/mysql \
-d mysql/enterprise-server:8.0
```

With this command, the MySQL Server is started with a Docker image of the MySQL Enterprise Edition, and the host directory */path-on-host-machine/datadir/* has been mounted onto the server's data directory (`/var/lib/mysql`) inside the server container. We also assume that, after the server has been started, the required privileges have also been set up for MySQL Enterprise Backup to access the server (see Grant MySQL Privileges to Backup Administrator, for details). Use the following steps to back up and restore a MySQL Server instance.

To back up a MySQL Server instance running in a Docker container using MySQL Enterprise Backup with Docker, follow the steps listed here:

1. On the same host where the MySQL Server container is running, start another container with an image of MySQL Enterprise Edition to perform a back up with the MySQL Enterprise Backup command `backup-to-image`. Provide access to the server's data directory using the bind mount we created in the last step. Also, mount a host directory (*/path-on-host-machine/backups/* in this example) onto the storage folder for backups in the container (`/data/backups` in the example) to persist the backups we are creating. Here is a sample command for this step, in which MySQL Enterprise Backup is started with a Docker image downloaded from My Oracle Support):

```
$> docker run \
--mount type=bind,src=/path-on-host-machine/datadir/,dst=/var/lib/mysql \
--mount type=bind,src=/path-on-host-machine/backups/,dst=/data/backups \
--rm mysql/enterprise-server:8.0 \
mysqlbackup -umysqlbackup -ppassword --backup-dir=/tmp/backup-tmp --with-timestamp \
--backup-image=/data/backups/db.mbi backup-to-image

[Entrypoint] MySQL Docker Image 8.0.11-1.1.5
MySQL Enterprise Backup version 8.0.11 Linux-4.1.12-61.1.16.el7uek.x86_64-x86_64 [2018-04-08  07:06:45]
Copyright (c) 2003, 2018, Oracle and/or its affiliates. All Rights Reserved.

180921 17:27:25 MAIN    INFO: A thread created with Id '140594390935680'
180921 17:27:25 MAIN    INFO: Starting with following command line ...
...

------------------------------------------------------------
   Parameters Summary
------------------------------------------------------------
```

```
   Start LSN                  : 29615616
   End LSN                    : 29651854
------------------------------------------------------------

mysqlbackup completed OK!
```

It is important to check the end of the output by `mysqlbackup` to make sure the backup has been completed successfully.

2. The container exits once the backup job is finished and, with the `--rm` option used to start it, it is removed after it exits. An image backup has been created, and can be found in the host directory mounted in the last step for storing backups, as shown here:

```
$> ls /tmp/backups
db.mbi
```

To restore a MySQL Server instance in a Docker container using MySQL Enterprise Backup with Docker, follow the steps listed here:

1. Stop the MySQL Server container, which also stops the MySQL Server running inside:

```
docker stop mysqlserver
```

2. On the host, delete all contents in the bind mount for the MySQL Server data directory:

```
rm -rf /path-on-host-machine/datadir/*
```

3. Start a container with an image of MySQL Enterprise Edition to perform the restore with the MySQL Enterprise Backup command `copy-back-and-apply-log`. Bind-mount the server's data directory and the storage folder for the backups, like what we did when we backed up the server:

```
$> docker run \
--mount type=bind,src=/path-on-host-machine/datadir/,dst=/var/lib/mysql \
--mount type=bind,src=/path-on-host-machine/backups/,dst=/data/backups \
--rm mysql/enterprise-server:8.0 \
mysqlbackup --backup-dir=/tmp/backup-tmp --with-timestamp \
--datadir=/var/lib/mysql --backup-image=/data/backups/db.mbi copy-back-and-apply-log

[Entrypoint] MySQL Docker Image 8.0.11-1.1.5
MySQL Enterprise Backup version 8.0.11 Linux-4.1.12-61.1.16.el7uek.x86_64-x86_64 [2018-04-08  07:06:
Copyright (c) 2003, 2018, Oracle and/or its affiliates. All Rights Reserved.

180921 22:06:52 MAIN    INFO: A thread created with Id '139768047519872'
180921 22:06:52 MAIN    INFO: Starting with following command line ...
...
180921 22:06:52 PCR1    INFO: We were able to parse ibbackup_logfile up to
          lsn 29680612.
180921 22:06:52 PCR1    INFO: Last MySQL binlog file position 0 155, file name binlog.000003
180921 22:06:52 PCR1    INFO: The first data file is '/var/lib/mysql/ibdata1'
                              and the new created log files are at '/var/lib/mysql'
180921 22:06:52 MAIN    INFO: No Keyring file to process.
180921 22:06:52 MAIN    INFO: Apply-log operation completed successfully.
180921 22:06:52 MAIN    INFO: Full Backup has been restored successfully.

mysqlbackup completed OK! with 3 warnings
```

The container exits once the backup job is finished and, with the `--rm` option used when starting it, it is removed after it exits.

4. Restart the server container, which also restarts the restored server, using the following command:

```
docker restart mysqlserver
```

Or, start a new MySQL Server on the restored data directory, as shown here:

```
docker run --name=mysqlserver2 \
--mount type=bind,src=/path-on-host-machine/datadir/,dst=/var/lib/mysql \
-d mysql/enterprise-server:8.0
```

Log on to the server to check that the server is running with the restored data.

Using `mysqldump` with Docker

Besides using MySQL Enterprise Backup to back up a MySQL Server running in a Docker container, you can perform a logical backup of your server by using the `mysqldump` utility, run inside a Docker container.

The following instructions assume that you already have a MySQL Server running in a Docker container and, when the container was first started, a host directory */path-on-host-machine/datadir/* has been mounted onto the server's data directory `/var/lib/mysql` (see bind-mounting a host directory on the data directory of the MySQL Server for details), which contains the Unix socket file by which `mysqldump` and `mysql` can connect to the server. We also assume that, after the server has been started, a user with the proper privileges (`admin` in this example) has been created, with which `mysqldump` can access the server. Use the following steps to back up and restore MySQL Server data:

Backing up MySQL Server data using `mysqldump` with Docker:

1. On the same host where the MySQL Server container is running, start another container with an image of MySQL Server to perform a backup with the `mysqldump` utility (see documentation of the utility for its functionality, options, and limitations). Provide access to the server's data directory by bind mounting */path-on-host-machine/datadir/*. Also, mount a host directory (*/path-on-host-machine/backups/* in this example) onto a storage folder for backups inside the container (`/data/backups` is used in this example) to persist the backups you are creating. Here is a sample command for backing up all databases on the server using this setup:

   ```
   $> docker run --entrypoint "/bin/sh" \
   --mount type=bind,src=/path-on-host-machine/datadir/,dst=/var/lib/mysql \
   --mount type=bind,src=/path-on-host-machine/backups/,dst=/data/backups \
   --rm mysql/mysql-server:8.0 \
   -c "mysqldump -uadmin --password='password' --all-databases > /data/backups/all-databases.sql"
   ```

   In the command, the `--entrypoint` option is used so that the system shell is invoked after the container is started, and the `-c` option is used to specify the `mysqldump` command to be run in the shell, whose output is redirected to the file `all-databases.sql` in the backup directory.

2. The container exits once the backup job is finished and, with the `--rm` option used to start it, it is removed after it exits. A logical backup been created, and can be found in the host directory mounted for storing the backup, as shown here:

   ```
   $> ls /path-on-host-machine/backups/
   all-databases.sql
   ```

Restoring MySQL Server data using `mysqldump` with Docker:

1. Make sure you have a MySQL Server running in a container, onto which you want your backed-up data to be restored.

2. Start a container with an image of MySQL Server to perform the restore with a `mysql` client. Bind-mount the server's data directory, as well as the storage folder that contains your backup:

   ```
   $> docker run  \
   --mount type=bind,src=/path-on-host-machine/datadir/,dst=/var/lib/mysql \
   --mount type=bind,src=/path-on-host-machine/backups/,dst=/data/backups \
   --rm mysql/mysql-server:8.0 \
   mysql -uadmin --password='password' -e "source /data/backups/all-databases.sql"
   ```

   The container exits once the backup job is finished and, with the `--rm` option used when starting it, it is removed after it exits.

3. Log on to the server to check that the restored data is now on the server.

Known Issues

- When using the server system variable `audit_log_file` to configure the audit log file name, use the `loose` option modifier with it; otherwise, Docker cannot start the server.

Docker Environment Variables

When you create a MySQL Server container, you can configure the MySQL instance by using the `--env` option (short form `-e`) and specifying one or more environment variables. No server initialization is performed if the mounted data directory is not empty, in which case setting any of these variables has no effect (see Persisting Data and Configuration Changes), and no existing contents of the directory, including server settings, are modified during container startup.

Environment variables which can be used to configure a MySQL instance are listed here:

- The boolean variables including `MYSQL_RANDOM_ROOT_PASSWORD`, `MYSQL_ONETIME_PASSWORD`, `MYSQL_ALLOW_EMPTY_PASSWORD`, and `MYSQL_LOG_CONSOLE` are made true by setting them with any strings of nonzero lengths. Therefore, setting them to, for example, “0”, “false”, or “no” does not make them false, but actually makes them true. This is a known issue.

- `MYSQL_RANDOM_ROOT_PASSWORD`: When this variable is true (which is its default state, unless `MYSQL_ROOT_PASSWORD` is set or `MYSQL_ALLOW_EMPTY_PASSWORD` is set to true), a random password for the server's root user is generated when the Docker container is started. The password is printed to `stdout` of the container and can be found by looking at the container’s log (see Starting a MySQL Server Instance).

- `MYSQL_ONETIME_PASSWORD`: When the variable is true (which is its default state, unless `MYSQL_ROOT_PASSWORD` is set or `MYSQL_ALLOW_EMPTY_PASSWORD` is set to true), the root user's password is set as expired and must be changed before MySQL can be used normally.

- `MYSQL_DATABASE`: This variable allows you to specify the name of a database to be created on image startup. If a user name and a password are supplied with `MYSQL_USER` and `MYSQL_PASSWORD`, the user is created and granted superuser access to this database (corresponding to `GRANT ALL`). The specified database is created by a CREATE DATABASE IF NOT EXIST statement, so that the variable has no effect if the database already exists.

- `MYSQL_USER`, `MYSQL_PASSWORD`: These variables are used in conjunction to create a user and set that user's password, and the user is granted superuser permissions for the database specified by the `MYSQL_DATABASE` variable. Both `MYSQL_USER` and `MYSQL_PASSWORD` are required for a user to be created—if any of the two variables is not set, the other is ignored. If both variables are set but `MYSQL_DATABASE` is not, the user is created without any privileges.

  Note

  There is no need to use this mechanism to create the root superuser, which is created by default with the password set by either one of the mechanisms discussed in the descriptions for `MYSQL_ROOT_PASSWORD` and `MYSQL_RANDOM_ROOT_PASSWORD`, unless `MYSQL_ALLOW_EMPTY_PASSWORD` is true.

- `MYSQL_ROOT_HOST`: By default, MySQL creates the `'root'@'localhost'` account. This account can only be connected to from inside the container as described in Connecting to MySQL Server from within the Container. To allow root connections from other hosts, set this environment variable. For example, the value `172.17.0.1`, which is the default Docker gateway IP, allows connections from the host machine that runs the container. The option accepts only one entry, but wildcards are allowed (for example, `MYSQL_ROOT_HOST=172.*.*.*` or `MYSQL_ROOT_HOST=%`).

- `MYSQL_LOG_CONSOLE`: When the variable is true (which is its default state for MySQL 8.0 server containers), the MySQL Server's error log is redirected to `stderr`, so that the error log goes into the Docker container's log and is viewable using the `docker logs` *`mysqld-container`* command.

Note

The variable has no effect if a server configuration file from the host has been mounted (see Persisting Data and Configuration Changes on bind-mounting a configuration file).

- `MYSQL_ROOT_PASSWORD`: This variable specifies a password that is set for the MySQL root account.

Warning

Setting the MySQL root user password on the command line is insecure. As an alternative to specifying the password explicitly, you can set the variable with a container file path for a password file, and then mount a file from your host that contains the password at the container file path. This is still not very secure, as the location of the password file is still exposed. It is preferable to use the default settings of `MYSQL_RANDOM_ROOT_PASSWORD` and `MYSQL_ONETIME_PASSWORD` both being true.

- `MYSQL_ALLOW_EMPTY_PASSWORD`. Set it to true to allow the container to be started with a blank password for the root user.

Warning

Setting this variable to true is insecure, because it is going to leave your MySQL instance completely unprotected, allowing anyone to gain complete superuser access. It is preferable to use the default settings of `MYSQL_RANDOM_ROOT_PASSWORD` and `MYSQL_ONETIME_PASSWORD` both being true.

2.5.6.3 Deploying MySQL on Windows and Other Non-Linux Platforms with Docker

Warning

The MySQL Docker images provided by Oracle are built specifically for Linux platforms. Other platforms are not supported, and users running the MySQL Docker images from Oracle on them are doing so at their own risk. This section discusses some known issues for the images when used on non-Linux platforms.

Known Issues for using the MySQL Server Docker images from Oracle on Windows include:

- If you are bind-mounting on the container's MySQL data directory (see Persisting Data and Configuration Changes for details), you have to set the location of the server socket file with the `--socket` option to somewhere outside of the MySQL data directory; otherwise, the server fails to start. This is because the way Docker for Windows handles file mounting does not allow a host file from being bind-mounted on the socket file.

2.5.7 Installing MySQL on Linux from the Native Software Repositories

Many Linux distributions include a version of the MySQL server, client tools, and development components in their native software repositories and can be installed with the platforms' standard package management systems. This section provides basic instructions for installing MySQL using those package management systems.

Important

Native packages are often several versions behind the currently available release. You are also normally unable to install development milestone releases (DMRs), since these are not usually made available in the native repositories.

> Before proceeding, we recommend that you check out the other installation options described in Section 2.5, “Installing MySQL on Linux”.

Distribution specific instructions are shown below:

- **Red Hat Linux, Fedora, CentOS**

> **Note**
>
> For a number of Linux distributions, you can install MySQL using the MySQL Yum repository instead of the platform's native software repository. See Section 2.5.1, “Installing MySQL on Linux Using the MySQL Yum Repository” for details.

For Red Hat and similar distributions, the MySQL distribution is divided into a number of separate packages, `mysql` for the client tools, `mysql-server` for the server and associated tools, and `mysql-libs` for the libraries. The libraries are required if you want to provide connectivity from different languages and environments such as Perl, Python and others.

To install, use the `yum` command to specify the packages that you want to install. For example:

```
#> yum install mysql mysql-server mysql-libs mysql-server
Loaded plugins: presto, refresh-packagekit
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package mysql.x86_64 0:5.1.48-2.fc13 set to be updated
---> Package mysql-libs.x86_64 0:5.1.48-2.fc13 set to be updated
---> Package mysql-server.x86_64 0:5.1.48-2.fc13 set to be updated
--> Processing Dependency: perl-DBD-MySQL for package: mysql-server-5.1.48-2.fc13.x86_64
--> Running transaction check
---> Package perl-DBD-MySQL.x86_64 0:4.017-1.fc13 set to be updated
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package               Arch          Version               Repository      Size
================================================================================
Installing:
 mysql                 x86_64        5.1.48-2.fc13         updates        889 k
 mysql-libs            x86_64        5.1.48-2.fc13         updates        1.2 M
 mysql-server          x86_64        5.1.48-2.fc13         updates        8.1 M
Installing for dependencies:
 perl-DBD-MySQL        x86_64        4.017-1.fc13          updates        136 k

Transaction Summary
================================================================================
Install       4 Package(s)
Upgrade       0 Package(s)

Total download size: 10 M
Installed size: 30 M
Is this ok [y/N]: y
Downloading Packages:
Setting up and reading Presto delta metadata
Processing delta metadata
Package(s) data still to download: 10 M
(1/4): mysql-5.1.48-2.fc13.x86_64.rpm                     | 889 kB     00:04
(2/4): mysql-libs-5.1.48-2.fc13.x86_64.rpm                | 1.2 MB     00:06
(3/4): mysql-server-5.1.48-2.fc13.x86_64.rpm              | 8.1 MB     00:40
(4/4): perl-DBD-MySQL-4.017-1.fc13.x86_64.rpm             | 136 kB     00:00
--------------------------------------------------------------------------------
Total                                          201 kB/s |  10 MB     00:52
Running rpm_check_debug
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing     : mysql-libs-5.1.48-2.fc13.x86_64                         1/4
```

```
  Installing     : mysql-5.1.48-2.fc13.x86_64                               2/4
  Installing     : perl-DBD-MySQL-4.017-1.fc13.x86_64                       3/4
  Installing     : mysql-server-5.1.48-2.fc13.x86_64                        4/4

Installed:
  mysql.x86_64 0:5.1.48-2.fc13            mysql-libs.x86_64 0:5.1.48-2.fc13
  mysql-server.x86_64 0:5.1.48-2.fc13

Dependency Installed:
  perl-DBD-MySQL.x86_64 0:4.017-1.fc13

Complete!
```

MySQL and the MySQL server should now be installed. A sample configuration file is installed into `/etc/my.cnf`. To start the MySQL server use `systemctl`:

```
$> systemctl start mysqld
```

The database tables are automatically created for you, if they do not already exist. You should, however, run `mysql_secure_installation` to set the root passwords on your server.

- **Debian, Ubuntu, Kubuntu**

Note

For supported Debian and Ubuntu versions, MySQL can be installed using the MySQL APT Repository instead of the platform's native software repository. See Section 2.5.2, “Installing MySQL on Linux Using the MySQL APT Repository” for details.

On Debian and related distributions, there are two packages for MySQL in their software repositories, `mysql-client` and `mysql-server`, for the client and server components respectively. You should specify an explicit version, for example `mysql-client-5.1`, to ensure that you install the version of MySQL that you want.

To download and install, including any dependencies, use the `apt-get` command, specifying the packages that you want to install.

Note

Before installing, make sure that you update your `apt-get` index files to ensure you are downloading the latest available version.

Note

The `apt-get` command installs a number of packages, including the MySQL server, in order to provide the typical tools and application environment. This can mean that you install a large number of packages in addition to the main MySQL package.

During installation, the initial database is created, and you are prompted for the MySQL root password (and confirmation). A configuration file is created in `/etc/mysql/my.cnf`. An init script is created in `/etc/init.d/mysql`.

The server should already be started. You can manually start and stop the server using:

```
#> service mysql [start|stop]
```

The service is automatically added to the 2, 3 and 4 run levels, with stop scripts in the single, shutdown and restart levels.

2.5.8 Installing MySQL on Linux with Juju

The Juju deployment framework supports easy installation and configuration of MySQL servers. For instructions, see https://jujucharms.com/mysql/.

2.5.9 Managing MySQL Server with systemd

If you install MySQL using an RPM or Debian package on the following Linux platforms, server startup and shutdown is managed by systemd:

- RPM package platforms:
  - Enterprise Linux variants version 7 and higher
  - SUSE Linux Enterprise Server 12 and higher
  - Fedora 29 and higher
- Debian family platforms:
  - Debian platforms
  - Ubuntu platforms

If you install MySQL from a generic binary distribution on a platform that uses systemd, you can manually configure systemd support for MySQL following the instructions provided in the post-installation setup section of the MySQL 8.0 Secure Deployment Guide.

If you install MySQL from a source distribution on a platform that uses systemd, obtain systemd support for MySQL by configuring the distribution using the `-DWITH_SYSTEMD=1` `CMake` option. See Section 2.8.7, “MySQL Source-Configuration Options”.

The following discussion covers these topics:

- Overview of systemd
- Configuring systemd for MySQL
- Configuring Multiple MySQL Instances Using systemd
- Migrating from mysqld_safe to systemd

Note

On platforms for which systemd support for MySQL is installed, scripts such as `mysqld_safe` and the System V initialization script are unnecessary and are not installed. For example, `mysqld_safe` can handle server restarts, but systemd provides the same capability, and does so in a manner consistent with management of other services rather than by using an application-specific program.

One implication of the non-use of `mysqld_safe` on platforms that use systemd for server management is that use of `[mysqld_safe]` or `[safe_mysqld]` sections in option files is not supported and might lead to unexpected behavior.

Because systemd has the capability of managing multiple MySQL instances on platforms for which systemd support for MySQL is installed, `mysqld_multi` and `mysqld_multi.server` are unnecessary and are not installed.

Overview of systemd

systemd provides automatic MySQL server startup and shutdown. It also enables manual server management using the `systemctl` command. For example:

```
$> systemctl {start|stop|restart|status} mysqld
```

Alternatively, use the `service` command (with the arguments reversed), which is compatible with System V systems:

```
$> service mysqld {start|stop|restart|status}
```

Note

For the `systemctl` command (and the alternative `service` command), if the MySQL service name is not `mysqld` then use the appropriate name. For example, use `mysql` rather than `mysqld` on Debian-based and SLES systems.

Support for systemd includes these files:

- `mysqld.service` (RPM platforms), `mysql.service` (Debian platforms): systemd service unit configuration file, with details about the MySQL service.
- `mysqld@.service` (RPM platforms), `mysql@.service` (Debian platforms): Like `mysqld.service` or `mysql.service`, but used for managing multiple MySQL instances.
- `mysqld.tmpfiles.d`: File containing information to support the `tmpfiles` feature. This file is installed under the name `mysql.conf`.
- `mysqld_pre_systemd` (RPM platforms), `mysql-system-start` (Debian platforms): Support script for the unit file. This script assists in creating the error log file only if the log location matches a pattern (`/var/log/mysql*.log` for RPM platforms, `/var/log/mysql/*.log` for Debian platforms). In other cases, the error log directory must be writable or the error log must be present and writable for the user running the `mysqld` process.

Configuring systemd for MySQL

To add or change systemd options for MySQL, these methods are available:

- Use a localized systemd configuration file.
- Arrange for systemd to set environment variables for the MySQL server process.
- Set the `MYSQLD_OPTS` systemd variable.

To use a localized systemd configuration file, create the `/etc/systemd/system/mysqld.service.d` directory if it does not exist. In that directory, create a file that contains a `[Service]` section listing the desired settings. For example:

```
[Service]
LimitNOFILE=max_open_files
Nice=nice_level
LimitCore=core_file_limit
Environment="LD_PRELOAD=/path/to/malloc/library"
Environment="TZ=time_zone_setting"
```

The discussion here uses `override.conf` as the name of this file. Newer versions of systemd support the following command, which opens an editor and permits you to edit the file:

```
systemctl edit mysqld  # RPM platforms
systemctl edit mysql   # Debian platforms
```

Whenever you create or change `override.conf`, reload the systemd configuration, then tell systemd to restart the MySQL service:

```
systemctl daemon-reload
systemctl restart mysqld  # RPM platforms
systemctl restart mysql   # Debian platforms
```

With systemd, the `override.conf` configuration method must be used for certain parameters, rather than settings in a `[mysqld]`, `[mysqld_safe]`, or `[safe_mysqld]` group in a MySQL option file:

- For some parameters, `override.conf` must be used because systemd itself must know their values and it cannot read MySQL option files to get them.
- Parameters that specify values otherwise settable only using options known to `mysqld_safe` must be specified using systemd because there is no corresponding `mysqld` parameter.

For additional information about using systemd rather than `mysqld_safe`, see Migrating from mysqld_safe to systemd.

You can set the following parameters in `override.conf`:

- To set the number of file descriptors available to the MySQL server, use `LimitNOFILE` in `override.conf` rather than the `open_files_limit` system variable for `mysqld` or `--open-files-limit` option for `mysqld_safe`.
- To set the maximum core file size, use `LimitCore` in `override.conf` rather than the `--core-file-size` option for `mysqld_safe`.
- To set the scheduling priority for the MySQL server, use `Nice` in `override.conf` rather than the `--nice` option for `mysqld_safe`.

Some MySQL parameters are configured using environment variables:

- `LD_PRELOAD`: Set this variable if the MySQL server should use a specific memory-allocation library.
- `NOTIFY_SOCKET`: This environment variable specifies the socket that `mysqld` uses to communicate notification of startup completion and service status change with systemd. It is set by systemd when the `mysqld` service is started. The `mysqld` service reads the variable setting and writes to the defined location.

  In MySQL 8.0, `mysqld` uses the `Type=notify` process startup type. (`Type=forking` was used in MySQL 5.7.) With `Type=notify`, systemd automatically configures a socket file and exports the path to the `NOTIFY_SOCKET` environment variable.
- `TZ`: Set this variable to specify the default time zone for the server.

There are multiple ways to specify environment variable values for use by the MySQL server process managed by systemd:

- Use `Environment` lines in the `override.conf` file. For the syntax, see the example in the preceding discussion that describes how to use this file.
- Specify the values in the `/etc/sysconfig/mysql` file (create the file if it does not exist). Assign values using the following syntax:

  ```
  LD_PRELOAD=/path/to/malloc/library
  TZ=time_zone_setting
  ```

  After modifying `/etc/sysconfig/mysql`, restart the server to make the changes effective:

  ```
  systemctl restart mysqld  # RPM platforms
  systemctl restart mysql   # Debian platforms
  ```

To specify options for `mysqld` without modifying systemd configuration files directly, set or unset the `MYSQLD_OPTS` systemd variable. For example:

```
systemctl set-environment MYSQLD_OPTS="--general_log=1"
systemctl unset-environment MYSQLD_OPTS
```

`MYSQLD_OPTS` can also be set in the `/etc/sysconfig/mysql` file.

After modifying the systemd environment, restart the server to make the changes effective:

```
systemctl restart mysqld  # RPM platforms
systemctl restart mysql   # Debian platforms
```

For platforms that use systemd, the data directory is initialized if empty at server startup. This might be a problem if the data directory is a remote mount that has temporarily disappeared: The mount point would appear to be an empty data directory, which then would be initialized as a new data directory. To suppress this automatic initialization behavior, specify the following line in the `/etc/sysconfig/mysql` file (create the file if it does not exist):

```
NO_INIT=true
```

Configuring Multiple MySQL Instances Using systemd

This section describes how to configure systemd for multiple instances of MySQL.

Note

Because systemd has the capability of managing multiple MySQL instances on platforms for which systemd support is installed, `mysqld_multi` and `mysqld_multi.server` are unnecessary and are not installed.

To use multiple-instance capability, modify the `my.cnf` option file to include configuration of key options for each instance. These file locations are typical:

- `/etc/my.cnf` or `/etc/mysql/my.cnf` (RPM platforms)
- `/etc/mysql/mysql.conf.d/mysqld.cnf` (Debian platforms)

For example, to manage two instances named `replica01` and `replica02`, add something like this to the option file:

RPM platforms:

```
[mysqld@replica01]
datadir=/var/lib/mysql-replica01
socket=/var/lib/mysql-replica01/mysql.sock
port=3307
log-error=/var/log/mysqld-replica01.log

[mysqld@replica02]
datadir=/var/lib/mysql-replica02
socket=/var/lib/mysql-replica02/mysql.sock
port=3308
log-error=/var/log/mysqld-replica02.log
```

Debian platforms:

```
[mysqld@replica01]
datadir=/var/lib/mysql-replica01
socket=/var/lib/mysql-replica01/mysql.sock
port=3307
log-error=/var/log/mysql/replica01.log

[mysqld@replica02]
datadir=/var/lib/mysql-replica02
socket=/var/lib/mysql-replica02/mysql.sock
port=3308
log-error=/var/log/mysql/replica02.log
```

The replica names shown here use `@` as the delimiter because that is the only delimiter supported by systemd.

Instances then are managed by normal systemd commands, such as:

```
systemctl start mysqld@replica01
systemctl start mysqld@replica02
```

To enable instances to run at boot time, do this:

```
systemctl enable mysqld@replica01
systemctl enable mysqld@replica02
```

Use of wildcards is also supported. For example, this command displays the status of all replica instances:

```
systemctl status 'mysqld@replica*'
```

For management of multiple MySQL instances on the same machine, systemd automatically uses a different unit file:

- `mysqld@.service` rather than `mysqld.service` (RPM platforms)
- `mysql@.service` rather than `mysql.service` (Debian platforms)

In the unit file, `%I` and `%i` reference the parameter passed in after the `@` marker and are used to manage the specific instance. For a command such as this:

```
systemctl start mysqld@replica01
```

systemd starts the server using a command such as this:

```
mysqld --defaults-group-suffix=@%I ...
```

The result is that the `[server]`, `[mysqld]`, and `[mysqld@replica01]` option groups are read and used for that instance of the service.

Note

On Debian platforms, AppArmor prevents the server from reading or writing `/var/lib/mysql-replica*`, or anything other than the default locations. To address this, you must customize or disable the profile in `/etc/apparmor.d/usr.sbin.mysqld`.

Note

On Debian platforms, the packaging scripts for MySQL uninstallation cannot currently handle `mysqld@` instances. Before removing or upgrading the package, you must stop any extra instances manually first.

Migrating from mysqld_safe to systemd

Because `mysqld_safe` is not installed on platforms that use systemd to manage MySQL, options previously specified for that program (for example, in an `[mysqld_safe]` or `[safe_mysqld]` option group) must be specified another way:

- Some `mysqld_safe` options are also understood by `mysqld` and can be moved from the `[mysqld_safe]` or `[safe_mysqld]` option group to the `[mysqld]` group. This does *not* include `--pid-file`, `--open-files-limit`, or `--nice`. To specify those options, use the `override.conf` systemd file, described previously.

  Note

  On systemd platforms, use of `[mysqld_safe]` and `[safe_mysqld]` option groups is not supported and may lead to unexpected behavior.

- For some `mysqld_safe` options, there are alternative `mysqld` procedures. For example, the `mysqld_safe` option for enabling `syslog` logging is `--syslog`, which is deprecated. To write error log output to the system log, use the instructions at Section 5.4.2.8, “Error Logging to the System Log”.
- `mysqld_safe` options not understood by `mysqld` can be specified in `override.conf` or environment variables. For example, with `mysqld_safe`, if the server should use a specific memory allocation library, this is specified using the `--malloc-lib` option. For installations that manage the server with systemd, arrange to set the `LD_PRELOAD` environment variable instead, as described previously.

2.6 Installing MySQL Using Unbreakable Linux Network (ULN)

Linux supports a number of different solutions for installing MySQL, covered in Section 2.5, “Installing MySQL on Linux”. One of the methods, covered in this section, is installing from Oracle's Unbreakable Linux Network (ULN). You can find information about Oracle Linux and ULN under http://linux.oracle.com/.

To use ULN, you need to obtain a ULN login and register the machine used for installation with ULN. This is described in detail in the ULN FAQ. The page also describes how to install and update packages.

Both Community and Commercial packages are supported, and each offers three MySQL channels:

- `Server`: MySQL Server
- `Connectors`: MySQL Connector/C++, MySQL Connector/J, MySQL Connector/ODBC, and MySQL Connector/Python.
- `Tools`: MySQL Router, MySQL Shell, and MySQL Workbench

The Community channels are available to all ULN users.

Accessing commercial MySQL ULN packages at oracle.linux.com requires you to provide a CSI with a valid commercial license for MySQL (Enterprise or Standard). As of this writing, valid purchases are 60944, 60945, 64911, and 64912. The appropriate CSI makes commercial MySQL subscription channels available in your ULN GUI interface.

Once MySQL has been installed using ULN, you can find information on starting and stopping the server, and more, at Section 2.5.7, “Installing MySQL on Linux from the Native Software Repositories”, particularly under Section 2.5.4, “Installing MySQL on Linux Using RPM Packages from Oracle”.

If you are changing your package source to use ULN and not changing which build of MySQL you are using, then back up your data, remove your existing binaries, and replace them with those from ULN. If a change of build is involved, we recommend the backup be a dump (`mysqldump` or `mysqlpump` or from MySQL Shell's backup utility) just in case you need to rebuild your data after the new binaries are in place. If this shift to ULN crosses a version boundary, consult this section before proceeding: Section 2.10, “Upgrading MySQL”.

Note

Oracle Linux 8 is supported as of MySQL 8.0.17, and the community Tools and Connectors channels were added with the MySQL 8.0.24 release.

2.7 Installing MySQL on Solaris

Note

MySQL 8.0 supports Solaris 11.4 and higher

MySQL on Solaris is available in a number of different formats.

- For information on installing using the native Solaris PKG format, see Section 2.7.1, “Installing MySQL on Solaris Using a Solaris PKG”.
- To use a standard `tar` binary installation, use the notes provided in Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”. Check the notes and hints at the end of this section for Solaris specific notes that you may need before or after installation.

Note

MySQL 5.7 has a dependency on the Oracle Developer Studio Runtime Libraries; but this does not apply to MySQL 8.0.

To obtain a binary MySQL distribution for Solaris in tarball or PKG format, https://dev.mysql.com/downloads/mysql/8.0.html.

Additional notes to be aware of when installing and using MySQL on Solaris:

- If you want to use MySQL with the `mysql` user and group, use the `groupadd` and `useradd` commands:

```
groupadd mysql
useradd -g mysql -s /bin/false mysql
```

- If you install MySQL using a binary tarball distribution on Solaris, because the Solaris `tar` cannot handle long file names, use GNU `tar` (`gtar`) to unpack the distribution. If you do not have GNU `tar` on your system, install it with the following command:

```
pkg install archiver/gnu-tar
```

- You should mount any file systems on which you intend to store `InnoDB` files with the `forcedirectio` option. (By default mounting is done without this option.) Failing to do so causes a significant drop in performance when using the `InnoDB` storage engine on this platform.

- If you would like MySQL to start automatically, you can copy `support-files/mysql.server` to `/etc/init.d` and create a symbolic link to it named `/etc/rc3.d/S99mysql.server`.

- If too many processes try to connect very rapidly to `mysqld`, you should see this error in the MySQL log:

```
Error in accept: Protocol error
```

  You might try starting the server with the `--back_log=50` option as a workaround for this.

- To configure the generation of core files on Solaris you should use the `coreadm` command. Because of the security implications of generating a core on a `setuid()` application, by default, Solaris does not support core files on `setuid()` programs. However, you can modify this behavior using `coreadm`. If you enable `setuid()` core files for the current user, they are generated using mode 600 and are owned by the superuser.

2.7.1 Installing MySQL on Solaris Using a Solaris PKG

You can install MySQL on Solaris using a binary package of the native Solaris PKG format instead of the binary tarball distribution.

Note

MySQL 5.7 has a dependency on the Oracle Developer Studio Runtime Libraries; but this does not apply to MySQL 8.0.

To use this package, download the corresponding `mysql-VERSION-solaris11-PLATFORM.pkg.gz` file, then uncompress it. For example:

```
$> gunzip mysql-8.0.32-solaris11-x86_64.pkg.gz
```

To install a new package, use `pkgadd` and follow the onscreen prompts. You must have root privileges to perform this operation:

```
$> pkgadd -d mysql-8.0.32-solaris11-x86_64.pkg

The following packages are available:
  1  mysql     MySQL Community Server (GPL)
               (i86pc) 8.0.32

Select package(s) you wish to process (or 'all' to process
all packages). (default: all) [?,??,q]:
```

The PKG installer installs all of the files and tools needed, and then initializes your database if one does not exist. To complete the installation, you should set the root password for MySQL as provided in the instructions at the end of the installation. Alternatively, you can run the `mysql_secure_installation` script that comes with the installation.

By default, the PKG package installs MySQL under the root path `/opt/mysql`. You can change only the installation root path when using `pkgadd`, which can be used to install MySQL in a different Solaris zone. If you need to install in a specific directory, use a binary `tar` file distribution.

The `pkg` installer copies a suitable startup script for MySQL into `/etc/init.d/mysql`. To enable MySQL to startup and shutdown automatically, you should create a link between this file and the init script directories. For example, to ensure safe startup and shutdown of MySQL you could use the following commands to add the right links:

```
$> ln /etc/init.d/mysql /etc/rc3.d/S91mysql
$> ln /etc/init.d/mysql /etc/rc0.d/K02mysql
```

To remove MySQL, the installed package name is `mysql`. You can use this in combination with the `pkgrm` command to remove the installation.

To upgrade when using the Solaris package file format, you must remove the existing installation before installing the updated package. Removal of the package does not delete the existing database information, only the server, binaries and support files. The typical upgrade sequence is therefore:

```
$> mysqladmin shutdown
$> pkgrm mysql
$> pkgadd -d mysql-8.0.32-solaris11-x86_64.pkg
$> mysqld_safe &
$> mysql_upgrade   # prior to MySQL 8.0.16 only
```

You should check the notes in Section 2.10, “Upgrading MySQL” before performing any upgrade.

2.8 Installing MySQL from Source

Building MySQL from the source code enables you to customize build parameters, compiler optimizations, and installation location. For a list of systems on which MySQL is known to run, see https://www.mysql.com/support/supportedplatforms/database.html.

Before you proceed with an installation from source, check whether Oracle produces a precompiled binary distribution for your platform and whether it works for you. We put a great deal of effort into ensuring that our binaries are built with the best possible options for optimal performance. Instructions for installing binary distributions are available in Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”.

If you are interested in building MySQL from a source distribution using build options the same as or similar to those use by Oracle to produce binary distributions on your platform, obtain a binary distribution, unpack it, and look in the `docs/INFO_BIN` file, which contains information about how that MySQL distribution was configured and compiled.

Warning

Building MySQL with nonstandard options may lead to reduced functionality, performance, or security.

The MySQL source code contains internal documentation written using Doxygen. The generated Doxygen content is available at https://dev.mysql.com/doc/index-other.html. It is also possible to generate this content locally from a MySQL source distribution using the instructions at Section 2.8.10, “Generating MySQL Doxygen Documentation Content”.

2.8.1 Source Installation Methods

There are two methods for installing MySQL from source:

- Use a standard MySQL source distribution. To obtain a standard distribution, see Section 2.1.3, “How to Get MySQL”. For instructions on building from a standard distribution, see Section 2.8.4, “Installing MySQL Using a Standard Source Distribution”.

  Standard distributions are available as compressed `tar` files, Zip archives, or RPM packages. Distribution files have names of the form `mysql-VERSION.tar.gz`, `mysql-VERSION.zip`, or `mysql-VERSION.rpm`, where `VERSION` is a number like `8.0.32`. File names for source distributions can be distinguished from those for precompiled binary distributions in that source distribution names are generic and include no platform name, whereas binary distribution names include a platform name indicating the type of system for which the distribution is intended (for example, `pc-linux-i686` or `winx64`).

- Use a MySQL development tree. For information on building from one of the development trees, see Section 2.8.5, “Installing MySQL Using a Development Source Tree”.

2.8.2 Source Installation Prerequisites

Installation of MySQL from source requires several development tools. Some of these tools are needed no matter whether you use a standard source distribution or a development source tree. Other tool requirements depend on which installation method you use.

To install MySQL from source, the following system requirements must be satisfied, regardless of installation method:

- `CMake`, which is used as the build framework on all platforms. `CMake` can be downloaded from http://www.cmake.org.
- A good `make` program. Although some platforms come with their own `make` implementations, it is highly recommended that you use GNU `make` 3.75 or higher. It may already be available on your system as `gmake`. GNU `make` is available from http://www.gnu.org/software/make/.
- As of MySQL 8.0.27, MySQL 8.0 source code permits use of C++17 features. To enable a good level of C++17 support across all supported platforms, the following minimum compiler versions apply.
  - Linux: GCC 7.1 or Clang 5
  - macOS: XCode 10
  - Solaris: GCC 10
  - Windows: Visual Studio 2019 Update 4
- The MySQL C API requires a C++ or C99 compiler to compile.
- An SSL library is required for support of encrypted connections, entropy for random number generation, and other encryption-related operations. By default, the build uses the OpenSSL library installed on the host system. To specify the library explicitly, use the `WITH_SSL` option when you invoke `CMake`. For additional information, see Section 2.8.6, “Configuring SSL Library Support”.
- The Boost C++ libraries are required to build MySQL (but not to use it). MySQL compilation requires a particular Boost version. Typically, that is the current Boost version, but if a specific MySQL source distribution requires a different version, the configuration process stops with a message indicating the Boost version that it requires. To obtain Boost and its installation instructions, visit the official site. After Boost is installed, tell the build system where the Boost files are located by defining the `WITH_BOOST` option when you invoke `CMake`. For example:

  ```
  cmake . -DWITH_BOOST=/usr/local/boost_version_number
  ```

  Adjust the path as necessary to match your installation.
- The ncurses library.

- Sufficient free memory. If you encounter problems such as “internal compiler error” when compiling large source files, it may be that you have too little memory. If compiling on a virtual machine, try increasing the memory allocation.
- Perl is needed if you intend to run test scripts. Most Unix-like systems include Perl. On Windows, you can use a version such as ActiveState Perl.

To install MySQL from a standard source distribution, one of the following tools is required to unpack the distribution file:

- For a `.tar.gz` compressed `tar` file: GNU `gunzip` to uncompress the distribution and a reasonable `tar` to unpack it. If your `tar` program supports the `z` option, it can both uncompress and unpack the file.

  GNU `tar` is known to work. The standard `tar` provided with some operating systems is not able to unpack the long file names in the MySQL distribution. You should download and install GNU `tar`, or if available, use a preinstalled version of GNU tar. Usually this is available as `gnutar`, `gtar`, or as `tar` within a GNU or Free Software directory, such as `/usr/sfw/bin` or `/usr/local/bin`. GNU `tar` is available from http://www.gnu.org/software/tar/.
- For a `.zip` Zip archive: `WinZip` or another tool that can read `.zip` files.
- For an `.rpm` RPM package: The `rpmbuild` program used to build the distribution unpacks it.

To install MySQL from a development source tree, the following additional tools are required:

- The Git revision control system is required to obtain the development source code. The GitHub Help provides instructions for downloading and installing Git on different platforms. MySQL officially joined GitHub in September, 2014. For more information about MySQL's move to GitHub, refer to the announcement on the MySQL Release Engineering blog: MySQL on GitHub
- `bison` 2.1 or higher, available from http://www.gnu.org/software/bison/. (Version 1 is no longer supported.) Use the latest version of `bison` where possible; if you experience problems, upgrade to a later version, rather than revert to an earlier one.

  `bison` is available from http://www.gnu.org/software/bison/. `bison` for Windows can be downloaded from http://gnuwin32.sourceforge.net/packages/bison.htm. Download the package labeled “Complete package, excluding sources”. On Windows, the default location for `bison` is the `C:\Program Files\GnuWin32` directory. Some utilities may fail to find `bison` because of the space in the directory name. Also, Visual Studio may simply hang if there are spaces in the path. You can resolve these problems by installing into a directory that does not contain a space (for example `C:\GnuWin32`).
- On Solaris Express, `m4` must be installed in addition to `bison`. `m4` is available from http://www.gnu.org/software/m4/.

Note

If you have to install any programs, modify your `PATH` environment variable to include any directories in which the programs are located. See Section 4.2.9, “Setting Environment Variables”.

If you run into problems and need to file a bug report, please use the instructions in Section 1.5, “How to Report Bugs or Problems”.

2.8.3 MySQL Layout for Source Installation

By default, when you install MySQL after compiling it from source, the installation step installs files under `/usr/local/mysql`. The component locations under the installation directory are the same as for binary distributions. See Table 2.3, “MySQL Installation Layout for Generic Unix/Linux Binary Package”, and Section 2.3.1, “MySQL Installation Layout on Microsoft Windows”. To configure

installation locations different from the defaults, use the options described at Section 2.8.7, “MySQL Source-Configuration Options”.

2.8.4 Installing MySQL Using a Standard Source Distribution

To install MySQL from a standard source distribution:

1. Verify that your system satisfies the tool requirements listed at Section 2.8.2, “Source Installation Prerequisites”.
2. Obtain a distribution file using the instructions in Section 2.1.3, “How to Get MySQL”.
3. Configure, build, and install the distribution using the instructions in this section.
4. Perform postinstallation procedures using the instructions in Section 2.9, “Postinstallation Setup and Testing”.

MySQL uses `CMake` as the build framework on all platforms. The instructions given here should enable you to produce a working installation. For additional information on using `CMake` to build MySQL, see How to Build MySQL Server with CMake.

If you start from a source RPM, use the following command to make a binary RPM that you can install. If you do not have `rpmbuild`, use `rpm` instead.

```
$> rpmbuild --rebuild --clean MySQL-VERSION.src.rpm
```

The result is one or more binary RPM packages that you install as indicated in Section 2.5.4, “Installing MySQL on Linux Using RPM Packages from Oracle”.

The sequence for installation from a compressed `tar` file or Zip archive source distribution is similar to the process for installing from a generic binary distribution (see Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”), except that it is used on all platforms and includes steps to configure and compile the distribution. For example, with a compressed `tar` file source distribution on Unix, the basic installation command sequence looks like this:

```
# Preconfiguration setup
$> groupadd mysql
$> useradd -r -g mysql -s /bin/false mysql
# Beginning of source-build specific instructions
$> tar zxvf mysql-VERSION.tar.gz
$> cd mysql-VERSION
$> mkdir bld
$> cd bld
$> cmake ..
$> make
$> make install
# End of source-build specific instructions
# Postinstallation setup
$> cd /usr/local/mysql
$> mkdir mysql-files
$> chown mysql:mysql mysql-files
$> chmod 750 mysql-files
$> bin/mysqld --initialize --user=mysql
$> bin/mysql_ssl_rsa_setup
$> bin/mysqld_safe --user=mysql &
# Next command is optional
$> cp support-files/mysql.server /etc/init.d/mysql.server
```

A more detailed version of the source-build specific instructions is shown following.

Note

The procedure shown here does not set up any passwords for MySQL accounts. After following the procedure, proceed to Section 2.9, “Postinstallation Setup and Testing”, for postinstallation setup and testing.

- Perform Preconfiguration Setup
- Obtain and Unpack the Distribution
- Configure the Distribution
- Build the Distribution
- Install the Distribution
- Perform Postinstallation Setup

Perform Preconfiguration Setup

On Unix, set up the `mysql` user and group that should be used to run and execute the MySQL server, and own the database directory. For details, see Create a mysql User and Group. Then perform the following steps as the `mysql` user, except as noted.

Obtain and Unpack the Distribution

Pick the directory under which you want to unpack the distribution and change location into it.

Obtain a distribution file using the instructions in Section 2.1.3, “How to Get MySQL”.

Unpack the distribution into the current directory:

- To unpack a compressed `tar` file, `tar` can uncompress and unpack the distribution if it has `z` option support:

  ```
  $> tar zxvf mysql-VERSION.tar.gz
  ```

  If your `tar` does not have `z` option support, use `gunzip` to unpack the distribution and `tar` to unpack it:

  ```
  $> gunzip < mysql-VERSION.tar.gz | tar xvf -
  ```

  Alternatively, `CMake` can uncompress and unpack the distribution:

  ```
  $> cmake -E tar zxvf mysql-VERSION.tar.gz
  ```

- To unpack a Zip archive, use `WinZip` or another tool that can read `.zip` files.

Unpacking the distribution file creates a directory named `mysql-VERSION`.

Configure the Distribution

Change location into the top-level directory of the unpacked distribution:

```
$> cd mysql-VERSION
```

Build outside of the source tree to keep the tree clean. If the top-level source directory is named `mysql-src` under your current working directory, you can build in a directory named `bld` at the same level. Create the directory and go there:

```
$> mkdir bld
$> cd bld
```

Configure the build directory. The minimum configuration command includes no options to override configuration defaults:

```
$> cmake ../mysql-src
```

The build directory needs not be outside the source tree. For example, you can build in a directory named `bld` under the top-level source tree. To do this, starting with `mysql-src` as your current working directory, create the directory `bld` and then go there:

```
$> mkdir bld
$> cd bld
```

Configure the build directory. The minimum configuration command includes no options to override configuration defaults:

```
$> cmake ..
```

If you have multiple source trees at the same level (for example, to build multiple versions of MySQL), the second strategy can be advantageous. The first strategy places all build directories at the same level, which requires that you choose a unique name for each. With the second strategy, you can use the same name for the build directory within each source tree. The following instructions assume this second strategy.

On Windows, specify the development environment. For example, the following commands configure MySQL for 32-bit or 64-bit builds, respectively:

```
$> cmake .. -G "Visual Studio 12 2013"
$> cmake .. -G "Visual Studio 12 2013 Win64"
```

On macOS, to use the Xcode IDE:

```
$> cmake .. -G Xcode
```

When you run `cmake`, you might want to add options to the command line. Here are some examples:

- `-DBUILD_CONFIG=mysql_release`: Configure the source with the same build options used by Oracle to produce binary distributions for official MySQL releases.
- `-DCMAKE_INSTALL_PREFIX=`*`dir_name`*: Configure the distribution for installation under a particular location.
- `-DCPACK_MONOLITHIC_INSTALL=1`: Cause `make package` to generate a single installation file rather than multiple files.
- `-DWITH_DEBUG=1`: Build the distribution with debugging support.

For a more extensive list of options, see Section 2.8.7, “MySQL Source-Configuration Options”.

To list the configuration options, use one of the following commands:

```
$> cmake .. -L   # overview
$> cmake .. -LH  # overview with help text
$> cmake .. -LAH # all params with help text
$> ccmake ..     # interactive display
```

If `CMake` fails, you might need to reconfigure by running it again with different options. If you do reconfigure, take note of the following:

- If `CMake` is run after it has previously been run, it may use information that was gathered during its previous invocation. This information is stored in `CMakeCache.txt`. When `CMake` starts, it looks for that file and reads its contents if it exists, on the assumption that the information is still correct. That assumption is invalid when you reconfigure.
- Each time you run `CMake`, you must run `make` again to recompile. However, you may want to remove old object files from previous builds first because they were compiled using different configuration options.

To prevent old object files or configuration information from being used, run these commands in the build directory on Unix before re-running `CMake`:

```
$> make clean
$> rm CMakeCache.txt
```

Or, on Windows:

```
$> devenv MySQL.sln /clean
$> del CMakeCache.txt
```

Before asking on the MySQL Community Slack, check the files in the `CMakeFiles` directory for useful information about the failure. To file a bug report, please use the instructions in Section 1.5, “How to Report Bugs or Problems”.

Build the Distribution

On Unix:

```
$> make
$> make VERBOSE=1
```

The second command sets `VERBOSE` to show the commands for each compiled source.

Use `gmake` instead on systems where you are using GNU `make` and it has been installed as `gmake`.

On Windows:

```
$> devenv MySQL.sln /build RelWithDebInfo
```

If you have gotten to the compilation stage, but the distribution does not build, see Section 2.8.8, “Dealing with Problems Compiling MySQL”, for help. If that does not solve the problem, please enter it into our bugs database using the instructions given in Section 1.5, “How to Report Bugs or Problems”. If you have installed the latest versions of the required tools, and they crash trying to process our configuration files, please report that also. However, if you get a `command not found` error or a similar problem for required tools, do not report it. Instead, make sure that all the required tools are installed and that your `PATH` variable is set correctly so that your shell can find them.

Install the Distribution

On Unix:

```
$> make install
```

This installs the files under the configured installation directory (by default, `/usr/local/mysql`). You might need to run the command as `root`.

To install in a specific directory, add a `DESTDIR` parameter to the command line:

```
$> make install DESTDIR="/opt/mysql"
```

Alternatively, generate installation package files that you can install where you like:

```
$> make package
```

This operation produces one or more `.tar.gz` files that can be installed like generic binary distribution packages. See Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”. If you run `CMake` with `-DCPACK_MONOLITHIC_INSTALL=1`, the operation produces a single file. Otherwise, it produces multiple files.

On Windows, generate the data directory, then create a `.zip` archive installation package:

```
$> devenv MySQL.sln /build RelWithDebInfo /project initial_database
$> devenv MySQL.sln /build RelWithDebInfo /project package
```

You can install the resulting `.zip` archive where you like. See Section 2.3.4, “Installing MySQL on Microsoft Windows Using a `noinstall` ZIP Archive”.

Perform Postinstallation Setup

The remainder of the installation process involves setting up the configuration file, creating the core databases, and starting the MySQL server. For instructions, see Section 2.9, “Postinstallation Setup and Testing”.

Note

The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in Section 2.9, “Postinstallation Setup and Testing”.

2.8.5 Installing MySQL Using a Development Source Tree

This section describes how to install MySQL from the latest development source code, which is hosted on GitHub. To obtain the MySQL Server source code from this repository hosting service, you can set up a local MySQL Git repository.

On GitHub, MySQL Server and other MySQL projects are found on the MySQL page. The MySQL Server project is a single repository that contains branches for several MySQL series.

MySQL officially joined GitHub in September, 2014. For more information about MySQL's move to GitHub, refer to the announcement on the MySQL Release Engineering blog: MySQL on GitHub

- Prerequisites for Installing from Development Source
- Setting Up a MySQL Git Repository

Prerequisites for Installing from Development Source

To install MySQL from a development source tree, your system must satisfy the tool requirements listed at Section 2.8.2, “Source Installation Prerequisites”.

Setting Up a MySQL Git Repository

To set up a MySQL Git repository on your machine:

1. Clone the MySQL Git repository to your machine. The following command clones the MySQL Git repository to a directory named `mysql-server`. The initial download may take some time to complete, depending on the speed of your connection.

```
~$ git clone https://github.com/mysql/mysql-server.git
Cloning into 'mysql-server'...
remote: Counting objects: 1198513, done.
remote: Total 1198513 (delta 0), reused 0 (delta 0), pack-reused 1198513
Receiving objects: 100% (1198513/1198513), 1.01 GiB | 7.44 MiB/s, done.
Resolving deltas: 100% (993200/993200), done.
Checking connectivity... done.
Checking out files: 100% (25510/25510), done.
```

2. When the clone operation completes, the contents of your local MySQL Git repository appear similar to the following:

```
~$ cd mysql-server
~/mysql-server$ ls
client             extra               mysys              storage
cmake              include             packaging          strings
CMakeLists.txt     INSTALL             plugin             support-files
components         libbinlogevents     README             testclients
config.h.cmake     libbinlogstandalone router             unittest
configure.cmake    libmysql            run_doxygen.cmake  utilities
Docs               libservices         scripts            VERSION
Doxyfile-ignored   LICENSE             share              vio
Doxyfile.in        man                 sql                win
doxygen_resources  mysql-test          sql-common
```

3. Use the `git branch -r` command to view the remote tracking branches for the MySQL repository.

```
~/mysql-server$ git branch -r
  origin/5.5
```

```
  origin/5.6
  origin/5.7
  origin/8.0
  origin/HEAD -> origin/8.0
  origin/cluster-7.2
  origin/cluster-7.3
  origin/cluster-7.4
  origin/cluster-7.5
  origin/cluster-7.6
```

4. To view the branch that is checked out in your local repository, issue the `git branch` command. When you clone the MySQL Git repository, the latest MySQL GA branch is checked out automatically. The asterisk identifies the active branch.

```
~/mysql-server$ git branch
* 8.0
```

5. To check out an earlier MySQL branch, run the `git checkout` command, specifying the branch name. For example, to check out the MySQL 5.7 branch:

```
~/mysql-server$ git checkout 5.7
Checking out files: 100% (9600/9600), done.
Branch 5.7 set up to track remote branch 5.7 from origin.
Switched to a new branch '5.7'
```

6. To obtain changes made after your initial setup of the MySQL Git repository, switch to the branch you want to update and issue the `git pull` command:

```
~/mysql-server$ git checkout 8.0
~/mysql-server$ git pull
```

   To examine the commit history, use the `git log` option:

```
~/mysql-server$ git log
```

   You can also browse commit history and source code on the GitHub MySQL site.

   If you see changes or code that you have a question about, ask on the MySQL Community Slack. For information about contributing a patch, see Contributing to MySQL Server.

7. After you have cloned the MySQL Git repository and have checked out the branch you want to build, you can build MySQL Server from the source code. Instructions are provided in Section 2.8.4, “Installing MySQL Using a Standard Source Distribution”, except that you skip the part about obtaining and unpacking the distribution.

   Be careful about installing a build from a distribution source tree on a production machine. The installation command may overwrite your live release installation. If you already have MySQL installed and do not want to overwrite it, run `CMake` with values for the `CMAKE_INSTALL_PREFIX`, `MYSQL_TCP_PORT`, and `MYSQL_UNIX_ADDR` options different from those used by your production server. For additional information about preventing multiple servers from interfering with each other, see Section 5.8, “Running Multiple MySQL Instances on One Machine”.

   Play hard with your new installation. For example, try to make new features crash. Start by running `make test`. See The MySQL Test Suite.

2.8.6 Configuring SSL Library Support

An SSL library is required for support of encrypted connections, entropy for random number generation, and other encryption-related operations.

If you compile MySQL from a source distribution, `CMake` configures the distribution to use the installed OpenSSL library by default.

To compile using OpenSSL, use this procedure:

1. Ensure that OpenSSL 1.0.1 or higher is installed on your system. If the installed OpenSSL version is lower than 1.0.1, `CMake` produces an error at MySQL configuration time. If it is necessary to obtain OpenSSL, visit http://www.openssl.org.

2. The `WITH_SSL` `CMake` option determines which SSL library to use for compiling MySQL (see Section 2.8.7, “MySQL Source-Configuration Options”). The default is `-DWITH_SSL=system`, which uses OpenSSL. To make this explicit, specify that option on the `CMake` command line. For example:

   ```
   cmake . -DWITH_SSL=system
   ```

   That command configures the distribution to use the installed OpenSSL library. Alternatively, to explicitly specify the path name to the OpenSSL installation, use the following syntax. This can be useful if you have multiple versions of OpenSSL installed, to prevent `CMake` from choosing the wrong one:

   ```
   cmake . -DWITH_SSL=path_name
   ```

   Alternative OpenSSL system packages are supported as of v8.0.30 by using *WITH_SSL=openssl11* on EL7 or *WITH_SSL=openssl3* on EL8. Authentication plugins, such as LDAP and Kerberos, are disabled as they do not support these alternative versions of OpenSSL.

3. Compile and install the distribution.

To check whether a `mysqld` server supports encrypted connections, examine the value of the `have_ssl` system variable:

```
mysql> SHOW VARIABLES LIKE 'have_ssl';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| have_ssl      | YES   |
+---------------+-------+
```

If the value is `YES`, the server supports encrypted connections. If the value is `DISABLED`, the server is capable of supporting encrypted connections but was not started with the appropriate `--ssl-xxx` options to enable encrypted connections to be used; see Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”.

2.8.7 MySQL Source-Configuration Options

The `CMake` program provides a great deal of control over how you configure a MySQL source distribution. Typically, you do this using options on the `CMake` command line. For information about options supported by `CMake`, run either of these commands in the top-level source directory:

```
cmake . -LH
ccmake .
```

You can also affect `CMake` using certain environment variables. See Section 4.9, “Environment Variables”.

For boolean options, the value may be specified as 1 or `ON` to enable the option, or as 0 or `OFF` to disable the option.

Many options configure compile-time defaults that can be overridden at server startup. For example, the `CMAKE_INSTALL_PREFIX`, `MYSQL_TCP_PORT`, and `MYSQL_UNIX_ADDR` options that configure the default installation base directory location, TCP/IP port number, and Unix socket file can be changed at server startup with the `--basedir`, `--port`, and `--socket` options for `mysqld`. Where applicable, configuration option descriptions indicate the corresponding `mysqld` startup option.

The following sections provide more information about `CMake` options.

- CMake Option Reference

CMake Option Reference

The following table shows the available `CMake` options. In the `Default` column, `PREFIX` stands for the value of the `CMAKE_INSTALL_PREFIX` option, which specifies the installation base directory. This value is used as the parent location for several of the installation subdirectories.

Table 2.14 MySQL Source-Configuration Option Reference (CMake)

Formats	Description	Default	Introduced	Removed
`ADD_GDB_INDEX`	Whether to enable generation of .gdb_index section in binaries		8.0.18	
`BUILD_CONFIG`	Use same build options as official releases			
`BUNDLE_RUNTIME_LIBRARIES`	Bundle runtime libraries with server MSI and Zip packages for Windows	`OFF`		
`CMAKE_BUILD_TYPE`	Type of build to produce	`RelWithDebInfo`		
`CMAKE_CXX_FLAGS`	Flags for C++ Compiler			
`CMAKE_C_FLAGS`	Flags for C Compiler			
`CMAKE_INSTALL_PREFIX`	Installation base directory	`/usr/local/mysql`		
`COMPILATION_COMMENT`	Comment about compilation environment			
`COMPILATION_COMMENT_SERVER`	Comment about compilation environment for use by mysqld		8.0.14	
`COMPRESS_DEBUG_SECTIONS`	Compress debug sections of binary executables	`OFF`	8.0.22	
`CPACK_MONOLITHIC_INSTALL`	Whether package build produces single file	`OFF`		
`DEFAULT_CHARSET`	The default server character set	`utf8mb4`		

Formats	Description	Default	Introduced	Removed
DEFAULT_COLLATION	The default server collation	utf8mb4_0900_ai_ci		
DISABLE_PSI_COND	Exclude Performance Schema condition instrumentation	OFF		
DISABLE_PSI_DATA_LOCK	Exclude the performance schema data lock instrumentation	OFF		
DISABLE_PSI_ERROR	Exclude the performance schema server error instrumentation	OFF		
DISABLE_PSI_FILE	Exclude Performance Schema file instrumentation	OFF		
DISABLE_PSI_IDLE	Exclude Performance Schema idle instrumentation	OFF		
DISABLE_PSI_MEMORY	Exclude Performance Schema memory instrumentation	OFF		
DISABLE_PSI_METADATA	Exclude Performance Schema metadata instrumentation	OFF		
DISABLE_PSI_MUTEX	Exclude Performance Schema mutex instrumentation	OFF		
DISABLE_PSI_PS	Exclude the performance schema prepared statements	OFF		
DISABLE_PSI_RWLOCK	Exclude Performance Schema rwlock instrumentation	OFF		
DISABLE_PSI_SOCKET	Exclude Performance Schema socket instrumentation	OFF		
DISABLE_PSI_SP	Exclude Performance Schema stored	OFF		

Formats	Description	Default	Introduced	Removed
	program instrumentation			
DISABLE_PSI_STAGE	Exclude Performance Schema stage instrumentation	OFF		
DISABLE_PSI_STATEMENT	Exclude Performance Schema statement instrumentation	OFF		
DISABLE_PSI_STATEMENT_DIGEST	Exclude Performance Schema statements_digest instrumentation	OFF		
DISABLE_PSI_TABLE	Exclude Performance Schema table instrumentation	OFF		
DISABLE_PSI_THREAD	Exclude the performance schema thread instrumentation	OFF		
DISABLE_PSI_TRANSACTION	Exclude the performance schema transaction instrumentation	OFF		
DISABLE_SHARED	Do not build shared libraries, compile position-dependent code	OFF		8.0.18
DOWNLOAD_BOOST	Whether to download the Boost library	OFF		
DOWNLOAD_BOOST_TIMEOUT	Timeout in seconds for downloading the Boost library	600		
ENABLED_LOCAL_INFILE	Whether to enable LOCAL for LOAD DATA	OFF		
ENABLED_PROFILING	Whether to enable query profiling code	ON		
ENABLE_DOWNLOADS	Whether to download optional files	OFF		8.0.26
ENABLE_EXPERIMENTAL_SYSVARS	Whether to enabled experimental InnoDB system variables	OFF		

Formats	Description	Default	Introduced	Removed
ENABLE_GCOV	Whether to include gcov support			
ENABLE_GPROF	Enable gprof (optimized Linux builds only)	OFF		
FORCE_COLORED_OUTPUT	Whether to colorize compiler output	OFF	8.0.33	
FORCE_INSOURCE_BUILD	Whether to force an in-source build	OFF	8.0.14	
FORCE_UNSUPPORTED_COMPILER	Whether to permit unsupported compiler	OFF		
FPROFILE_GENERATE	Whether to generate profile guided optimization data	OFF	8.0.19	
FPROFILE_USE	Whether to use profile guided optimization data	OFF	8.0.19	
HAVE_PSI_MEMORY_INTERFACE	Enable performance schema memory tracing module for memory allocation functions used in dynamic storage of over-aligned types	OFF	8.0.26	
IGNORE_AIO_CHECK	With -DBUILD_CONFIG=mysql_release, ignore libaio check	OFF		
INSTALL_BINDIR	User executables directory	PREFIX/bin		
INSTALL_DOCDIR	Documentation directory	PREFIX/docs		
INSTALL_DOCREADMEDIR	README file directory	PREFIX		
INSTALL_INCLUDEDIR	Header file directory	PREFIX/include		
INSTALL_INFODIR	Info file directory	PREFIX/docs		
INSTALL_LAYOUT	Select predefined installation layout	STANDALONE		
INSTALL_LIBDIR	Library file directory	PREFIX/lib		
INSTALL_MANDIR	Manual page directory	PREFIX/man		
INSTALL_MYSQLKEYRINGDIR	Directory for keyring_file plugin data file	platform specific		

Formats	Description	Default	Introduced	Removed
INSTALL_MYSQLSHAREDIR	Shared data directory	PREFIX/share		
INSTALL_MYSQLTESTDIR	mysql-test directory	PREFIX/mysql-test		
INSTALL_PKGCONFIGDIR	Directory for mysqlclient.pc pkg-config file	INSTALL_LIBDIR/pkgconfig		
INSTALL_PLUGINDIR	Plugin directory	PREFIX/lib/plugin		
INSTALL_PRIV_LIBDIR	Installation private library directory		8.0.18	
INSTALL_SBINDIR	Server executable directory	PREFIX/bin		
INSTALL_SECURE_FILE_PRIVDIR	secure_file_priv default value	platform specific		
INSTALL_SHAREDIR	aclocal/mysql.m4 installation directory	PREFIX/share		
INSTALL_STATIC_LIBRARIES	Whether to install static libraries	ON		
INSTALL_SUPPORTFILESDIR	Extra support files directory	PREFIX/support-files		
LINK_RANDOMIZE	Whether to randomize order of symbols in mysqld binary	OFF		
LINK_RANDOMIZE_SEED	Seed value for LINK_RANDOMIZE option	mysql		
MAX_INDEXES	Maximum indexes per table	64		
MEMCACHED_HOME	Path to memcached; obsolete	[none]		8.0.23
MSVC_CPPCHECK	Enable MSVC code analysis.	OFF	8.0.33	
MUTEX_TYPE	InnoDB mutex type	event		
MYSQLX_TCP_PORT	TCP/IP port number used by X Plugin	33060		
MYSQLX_UNIX_ADDR	Unix socket file used by X Plugin	/tmp/mysqlx.sock		
MYSQL_DATADIR	Data directory			
MYSQL_MAINTAINER_MODE	Whether to enable MySQL maintainer-specific development environment	OFF		

Formats	Description	Default	Introduced	Removed
MYSQL_PROJECT_NAME	Windows/macOS project name	MySQL		
MYSQL_TCP_PORT	TCP/IP port number	3306		
MYSQL_UNIX_ADDR	Unix socket file	/tmp/mysql.sock		
NDB_UTILS_LINK_DYNAMIC	Cause NDB tools to be dynamically linked to ndbclient		8.0.22	
ODBC_INCLUDES	ODBC includes directory			
ODBC_LIB_DIR	ODBC library directory			
OPTIMIZER_TRACE	Whether to support optimizer tracing			
REPRODUCIBLE_BUILD	Take extra care to create a build result independent of build location and time			
SHOW_SUPPRESSED_COMPILER_WARNINGS	Whether to show suppressed compiler warnings, and without failing with -Werror.	OFF	8.0.30	
SYSCONFDIR	Option file directory			
SYSTEMD_PID_DIR	Directory for PID file under systemd	/var/run/mysqld		
SYSTEMD_SERVICE_NAME	Name of MySQL service under systemd	mysqld		
TMPDIR	tmpdir default value			
USE_LD_GOLD	Whether to use GNU gold linker	ON		
USE_LD_LLD	Whether to use llvm lld linker	ON	8.0.16	
WIN_DEBUG_NO_INLINE	Whether to disable function inlining	OFF		
WITHOUT_SERVER	Do not build the server	OFF		
WITHOUT_xxx_STORAGE_ENGINE	Exclude storage engine xxx from build			
WITH_ANT	Path to Ant for building GCS Java wrapper			
WITH_ASAN	Enable AddressSanitizer	OFF		

Formats	Description	Default	Introduced	Removed
WITH_ASAN_SCOPE	Enable AddressSanitizer -fsanitize-address-use-after-scope Clang flag	OFF		
WITH_AUTHENTICATION_CLIENT_PLUGINS	Enabled automatically if any corresponding server authentication plugins are built		8.0.26	
WITH_AUTHENTICATION_LDAP	Whether to report error if LDAP authentication plugins cannot be built	OFF		
WITH_AUTHENTICATION_PAM	Build PAM authentication plugin	OFF		
WITH_AWS_SDK	Location of Amazon Web Services software development kit			
WITH_BOOST	The location of the Boost library sources			
WITH_BUILD_ID	On Linux systems, generate a unique build ID	ON	8.0.31	
WITH_BUNDLED_LIBEVENT	Use bundled libevent when building ndbmemcache; obsolete	ON		8.0.23
WITH_BUNDLED_MEMCACHED	Use bundled memcached when building ndbmemcache; obsolete	ON		8.0.23
WITH_CLASSPATH	Classpath to use when building MySQL Cluster Connector for Java. Default is an empty string.			
WITH_CLIENT_PROTOCOL_TRACING	Build client-side protocol tracing framework	ON		
WITH_CURL	Location of curl library			

Formats	Description	Default	Introduced	Removed
WITH_DEBUG	Whether to include debugging support	OFF		
WITH_DEFAULT_COMPILER_OPTIONS	Whether to use default compiler options	ON		
WITH_DEFAULT_FEATURE_SET	Whether to use default feature set	ON		8.0.22
WITH_DEVELOPER_ENTITLEMENTS	Whether to add the 'get-task-allow' entitlement to all executables on macOS to generate a core dump in the event of an unexpected server halt	OFF	8.0.30	
WITH_EDITLINE	Which libedit/ editline library to use	bundled		
WITH_ERROR_INSERT	Enable error injection in the NDB storage engine. Should not be used for building binaries intended for production.	OFF		
WITH_FIDO	Type of FIDO library support	bundled	8.0.27	
WITH_GMOCK	Path to googlemock distribution			8.0.26
WITH_ICU	Type of ICU support	bundled		
WITH_INNODB_EXTRA_DEBUG	Whether to include extra debugging support for InnoDB.	OFF		
WITH_INNODB_MEMCACHED	Whether to generate memcached shared libraries.	OFF		
WITH_JEMALLOC	Whether to link with -ljemalloc	OFF	8.0.16	
WITH_KEYRING_TEST	Build the keyring test program	OFF		
WITH_LIBEVENT	Which libevent library to use	bundled		
WITH_LIBWRAP	Whether to include libwrap (TCP wrappers) support	OFF		

Formats	Description	Default	Introduced	Removed
WITH_LOCK_ORDER	Whether to enable LOCK_ORDER tooling	OFF	8.0.17	
WITH_LSAN	Whether to run LeakSanitizer, without AddressSanitizer	OFF	8.0.16	
WITH_LTO	Enable link-time optimizer	OFF	8.0.13	
WITH_LZ4	Type of LZ4 library support	bundled		
WITH_LZMA	Type of LZMA library support	bundled		8.0.16
WITH_MECAB	Compiles MeCab			
WITH_MSAN	Enable MemorySanitizer	OFF		
WITH_MSCRT_DEBUG	Enable Visual Studio CRT memory leak tracing	OFF		
WITH_MYSQLX	Whether to disable X Protocol	ON		
WITH_NDB	Build MySQL NDB Cluster	OFF	8.0.31	
WITH_NDBAPI_EXAMPLES	Build API example programs	OFF		
WITH_NDBCLUSTER	Build the NDB storage engine	OFF		
WITH_NDBCLUSTER_STORAGE_ENGINE	For internal use; may not work as expected in all circumstances; users should employ WITH_NDBCLUSTER instead	ON		
WITH_NDBMTD	Build multithreaded data node.	ON		
WITH_NDB_DEBUG	Produce a debug build for testing or troubleshooting.	OFF		
WITH_NDB_JAVA	Enable building of Java and ClusterJ support. Enabled by default. Supported in MySQL Cluster only.	ON		
WITH_NDB_PORT	Default port used by a management	[none]		

Formats	Description	Default	Introduced	Removed
	server built with this option. If this option was not used to build it, the management server's default port is 1186.			
WITH_NDB_TEST	Include NDB API test programs.	OFF		
WITH_NUMA	Set NUMA memory allocation policy			
WITH_PACKAGE_FLAGS	For flags typically used for RPM/DEB packages, whether to add them to standalone builds on those platforms		8.0.26	
WITH_PLUGIN_NDBCLUSTER	For internal use; may not work as expected in all circumstances. Users should employ WITH_NDBCLUSTER or WITH_NDB instead		8.0.13	8.0.31
WITH_PROTOBUF	Which Protocol Buffers package to use	bundled		
WITH_RAPID	Whether to build rapid development cycle plugins	ON		
WITH_RAPIDJSON	Type of RapidJSON support	bundled	8.0.13	
WITH_RE2	Type of RE2 library support	bundled		8.0.18
WITH_ROUTER	Whether to build MySQL Router	ON	8.0.16	
WITH_SSL	Type of SSL support	system		
WITH_SYSTEMD	Enable installation of systemd support files	OFF		
WITH_SYSTEMD_DEBUG	Enable additional systemd debug information	OFF	8.0.22	
WITH_SYSTEM_LIBS	Set system value of library options not set explicitly	OFF		

Formats	Description	Default	Introduced	Removed
WITH_TCMALLOC	Whether to link with -ltcmalloc	OFF	8.0.22	
WITH_TEST_TRACE_PLUGIN	Build test protocol trace plugin	OFF		
WITH_TSAN	Enable ThreadSanitizer	OFF		
WITH_UBSAN	Enable Undefined Behavior Sanitizer	OFF		
WITH_UNIT_TESTS	Compile MySQL with unit tests	ON		
WITH_UNIXODBC	Enable unixODBC support	OFF		
WITH_VALGRIND	Whether to compile in Valgrind header files	OFF		
WITH_WIN_JEMALLOC	Path to directory containing jemalloc.dll		8.0.29	
WITH_ZLIB	Type of zlib support	bundled		
WITH_ZSTD	Type of zstd support	bundled	8.0.18	
WITH_xxx_STORAGE_ENGINE	Compile storage engine xxx statically into server			

General Options

- -DBUILD_CONFIG=mysql_release

  This option configures a source distribution with the same build options used by Oracle to produce binary distributions for official MySQL releases.

- -DWITH_BUILD_ID=*bool*

  On Linux systems, generates a unique build ID which is used as the value of the build_id system variable and written to the MySQL server log on startup. Set this option to OFF to disable this feature.

  Added in MySQL 8.0.31; has no effect on platforms other than Linux.

- -DBUNDLE_RUNTIME_LIBRARIES=*bool*

  Whether to bundle runtime libraries with server MSI and Zip packages for Windows.

- -DCMAKE_BUILD_TYPE=*type*

  The type of build to produce:

  - RelWithDebInfo: Enable optimizations and generate debugging information. This is the default MySQL build type.
  - Release: Enable optimizations but omit debugging information to reduce the build size. This build type was added in MySQL 8.0.13.

  - `Debug`: Disable optimizations and generate debugging information. This build type is also used if the `WITH_DEBUG` option is enabled. That is, `-DWITH_DEBUG=1` has the same effect as `-DCMAKE_BUILD_TYPE=Debug`.

- `-DCPACK_MONOLITHIC_INSTALL=bool`

  This option affects whether the `make package` operation produces multiple installation package files or a single file. If disabled, the operation produces multiple installation package files, which may be useful if you want to install only a subset of a full MySQL installation. If enabled, it produces a single file for installing everything.

- `-DFORCE_INSOURCE_BUILD=bool`

  Defines whether to force an in-source build. Out-of-source builds are recommended, as they permit multiple builds from the same source, and cleanup can be performed quickly by removing the build directory. To force an in-source build, invoke `CMake` with `-DFORCE_INSOURCE_BUILD=ON`.

- `-DFORCE_COLORED_OUTPUT=bool`

  Defines whether to enable colorized compiler output for gcc and clang when compiling on the command line. Defaults to OFF.

Installation Layout Options

The `CMAKE_INSTALL_PREFIX` option indicates the base installation directory. Other options with names of the form `INSTALL_xxx` that indicate component locations are interpreted relative to the prefix and their values are relative pathnames. Their values should not include the prefix.

- `-DCMAKE_INSTALL_PREFIX=dir_name`

  The installation base directory.

  This value can be set at server startup with the `--basedir` option.

- `-DINSTALL_BINDIR=dir_name`

  Where to install user programs.

- `-DINSTALL_DOCDIR=dir_name`

  Where to install documentation.

- `-DINSTALL_DOCREADMEDIR=dir_name`

  Where to install `README` files.

- `-DINSTALL_INCLUDEDIR=dir_name`

  Where to install header files.

- `-DINSTALL_INFODIR=dir_name`

  Where to install Info files.

- `-DINSTALL_LAYOUT=name`

  Select a predefined installation layout:

  - `STANDALONE`: Same layout as used for `.tar.gz` and `.zip` packages. This is the default.
  - `RPM`: Layout similar to RPM packages.
  - `SVR4`: Solaris package layout.

- `DEB`: DEB package layout (experimental).

You can select a predefined layout but modify individual component installation locations by specifying other options. For example:

```
cmake . -DINSTALL_LAYOUT=SVR4 -DMYSQL_DATADIR=/var/mysql/data
```

The `INSTALL_LAYOUT` value determines the default value of the `secure_file_priv`, `keyring_encrypted_file_data`, and `keyring_file_data` system variables. See the descriptions of those variables in Section 5.1.8, “Server System Variables”, and Section 6.4.4.19, “Keyring System Variables”.

- `-DINSTALL_LIBDIR=`*`dir_name`*

  Where to install library files.

- `-DINSTALL_MANDIR=`*`dir_name`*

  Where to install manual pages.

- `-DINSTALL_MYSQLKEYRINGDIR=`*`dir_path`*

  The default directory to use as the location of the `keyring_file` plugin data file. The default value is platform specific and depends on the value of the `INSTALL_LAYOUT` `CMake` option; see the description of the `keyring_file_data` system variable in Section 5.1.8, “Server System Variables”.

- `-DINSTALL_MYSQLSHAREDIR=`*`dir_name`*

  Where to install shared data files.

- `-DINSTALL_MYSQLTESTDIR=`*`dir_name`*

  Where to install the `mysql-test` directory. To suppress installation of this directory, explicitly set the option to the empty value (`-DINSTALL_MYSQLTESTDIR=`).

- `-DINSTALL_PKGCONFIGDIR=`*`dir_name`*

  The directory in which to install the `mysqlclient.pc` file for use by `pkg-config`. The default value is `INSTALL_LIBDIR/pkgconfig`, unless `INSTALL_LIBDIR` ends with `/mysql`, in which case that is removed first.

- `-DINSTALL_PLUGINDIR=`*`dir_name`*

  The location of the plugin directory.

  This value can be set at server startup with the `--plugin_dir` option.

- `-DINSTALL_PRIV_LIBDIR=`*`dir_name`*

  The location of the dynamic library directory.

  Default locations: RPM = `/usr/lib64/mysql/private/`, DEB = `/usr/lib/mysql/private/`, and TAR = `lib/private/`.

  This option was added in MySQL 8.0.18.

  For Protobuf: Because this is a private location, loader (such as ld-linux.so on Linux) may not find the `libprotobuf.so` files without help. To guide loader, `RPATH` with value *$ORIGIN/../$INSTALL_PRIV_LIBDIR* is added to mysqld and mysqlxtest. This works for most cases but when using the Resource Group feature, *mysqld* is *setsuid* and then loader ignores `RPATH` which contains `$ORIGIN`. To overcome this, an explicit full path to the directory is set in DEB and RPM variants of

mysqld, as the target destination is known. For tarball installs, patching of mysqld with a tool like `patchelf` is required.

- `-DINSTALL_SBINDIR=dir_name`

  Where to install the `mysqld` server.

- `-DINSTALL_SECURE_FILE_PRIVDIR=dir_name`

  The default value for the `secure_file_priv` system variable. The default value is platform specific and depends on the value of the `INSTALL_LAYOUT` `CMake` option; see the description of the `secure_file_priv` system variable in Section 5.1.8, “Server System Variables”.

- `-DINSTALL_SHAREDIR=dir_name`

  Where to install `aclocal/mysql.m4`.

- `-DINSTALL_STATIC_LIBRARIES=bool`

  Whether to install static libraries. The default is `ON`. If set to `OFF`, these libraries are not installed: `libmysqlclient.a`, `libmysqlservices.a`.

- `-DINSTALL_SUPPORTFILESDIR=dir_name`

  Where to install extra support files.

- `-DLINK_RANDOMIZE=bool`

  Whether to randomize the order of symbols in the `mysqld` binary. The default is `OFF`. This option should be enabled only for debugging purposes.

- `-DLINK_RANDOMIZE_SEED=val`

  Seed value for the `LINK_RANDOMIZE` option. The value is a string. The default is `mysql`, an arbitrary choice.

- `-DMYSQL_DATADIR=dir_name`

  The location of the MySQL data directory.

  This value can be set at server startup with the `--datadir` option.

- `-DODBC_INCLUDES=dir_name`

  The location of the ODBC includes directory, and may be used while configuring Connector/ODBC.

- `-DODBC_LIB_DIR=dir_name`

  The location of the ODBC library directory, and may be used while configuring Connector/ODBC.

- `-DSYSCONFDIR=dir_name`

  The default `my.cnf` option file directory.

  This location cannot be set at server startup, but you can start the server with a given option file using the `--defaults-file=file_name` option, where *`file_name`* is the full path name to the file.

- `-DSYSTEMD_PID_DIR=dir_name`

  The name of the directory in which to create the PID file when MySQL is managed by systemd. The default is `/var/run/mysqld`; this might be changed implicitly according to the `INSTALL_LAYOUT` value.

This option is ignored unless `WITH_SYSTEMD` is enabled.

- `-DSYSTEMD_SERVICE_NAME=name`

  The name of the MySQL service to use when MySQL is managed by systemd. The default is `mysqld`; this might be changed implicitly according to the `INSTALL_LAYOUT` value.

  This option is ignored unless `WITH_SYSTEMD` is enabled.

- `-DTMPDIR=dir_name`

  The default location to use for the `tmpdir` system variable. If unspecified, the value defaults to `P_tmpdir` in `<stdio.h>`.

Storage Engine Options

Storage engines are built as plugins. You can build a plugin as a static module (compiled into the server) or a dynamic module (built as a dynamic library that must be installed into the server using the `INSTALL PLUGIN` statement or the `--plugin-load` option before it can be used). Some plugins might not support static or dynamic building.

The `InnoDB`, `MyISAM`, `MERGE`, `MEMORY`, and `CSV` engines are mandatory (always compiled into the server) and need not be installed explicitly.

To compile a storage engine statically into the server, use `-DWITH_engine_STORAGE_ENGINE=1`. Some permissible *`engine`* values are `ARCHIVE`, `BLACKHOLE`, `EXAMPLE`, and `FEDERATED`. Examples:

```
-DWITH_ARCHIVE_STORAGE_ENGINE=1
-DWITH_BLACKHOLE_STORAGE_ENGINE=1
```

To build MySQL with support for NDB Cluster, use the `WITH_NDB` option. (*NDB 8.0.30 and earlier*: Use `WITH_NDBCLUSTER`.)

Note

It is not possible to compile without Performance Schema support. If it is desired to compile without particular types of instrumentation, that can be done with the following `CMake` options:

```
DISABLE_PSI_COND
DISABLE_PSI_DATA_LOCK
DISABLE_PSI_ERROR
DISABLE_PSI_FILE
DISABLE_PSI_IDLE
DISABLE_PSI_MEMORY
DISABLE_PSI_METADATA
DISABLE_PSI_MUTEX
DISABLE_PSI_PS
DISABLE_PSI_RWLOCK
DISABLE_PSI_SOCKET
DISABLE_PSI_SP
DISABLE_PSI_STAGE
DISABLE_PSI_STATEMENT
DISABLE_PSI_STATEMENT_DIGEST
DISABLE_PSI_TABLE
DISABLE_PSI_THREAD
DISABLE_PSI_TRANSACTION
```

For example, to compile without mutex instrumentation, configure MySQL using the `-DDISABLE_PSI_MUTEX=1` option.

To exclude a storage engine from the build, use `-DWITH_engine_STORAGE_ENGINE=0`. Examples:

```
-DWITH_ARCHIVE_STORAGE_ENGINE=0
-DWITH_EXAMPLE_STORAGE_ENGINE=0
-DWITH_FEDERATED_STORAGE_ENGINE=0
```

It is also possible to exclude a storage engine from the build using `-DWITHOUT_engine_STORAGE_ENGINE=1` (but `-DWITH_engine_STORAGE_ENGINE=0` is preferred). Examples:

```
-DWITHOUT_ARCHIVE_STORAGE_ENGINE=1
-DWITHOUT_EXAMPLE_STORAGE_ENGINE=1
-DWITHOUT_FEDERATED_STORAGE_ENGINE=1
```

If neither `-DWITH_engine_STORAGE_ENGINE` nor `-DWITHOUT_engine_STORAGE_ENGINE` are specified for a given storage engine, the engine is built as a shared module, or excluded if it cannot be built as a shared module.

Feature Options

- `-DADD_GDB_INDEX=bool`

  This option determines whether to enable generation of a `.gdb_index` section in binaries, which makes loading them in a debugger faster. The option is disabled by default. `lld` linker is used, and is disabled by It has no effect if a linker other than `lld` or GNU `gold` is used.

  This option was added in MySQL 8.0.18.

- `-DCOMPILATION_COMMENT=string`

  A descriptive comment about the compilation environment. As of MySQL 8.0.14, `mysqld` uses `COMPILATION_COMMENT_SERVER`. Other programs continue to use `COMPILATION_COMMENT`.

- `-DCOMPRESS_DEBUG_SECTIONS=bool`

  Whether to compress the debug sections of binary executables (Linux only). Compressing executable debug sections saves space at the cost of extra CPU time during the build process.

  The default is `OFF`. If this option is not set explicitly but the `COMPRESS_DEBUG_SECTIONS` environment variable is set, the option takes its value from that variable.

  This option was added in MySQL 8.0.22.

- `-DCOMPILATION_COMMENT_SERVER=string`

  A descriptive comment about the compilation environment for use by `mysqld` (for example, to set the `version_comment` system variable). This option was added in MySQL 8.0.14. Prior to 8.0.14, the server uses `COMPILATION_COMMENT`.

- `-DDEFAULT_CHARSET=charset_name`

  The server character set. By default, MySQL uses the `utf8mb4` character set.

  `charset_name` may be one of `binary`, `armscii8`, `ascii`, `big5`, `cp1250`, `cp1251`, `cp1256`, `cp1257`, `cp850`, `cp852`, `cp866`, `cp932`, `dec8`, `eucjpms`, `euckr`, `gb2312`, `gbk`, `geostd8`, `greek`, `hebrew`, `hp8`, `keybcs2`, `koi8r`, `koi8u`, `latin1`, `latin2`, `latin5`, `latin7`, `macce`, `macroman`, `sjis`, `swe7`, `tis620`, `ucs2`, `ujis`, `utf8mb3`, `utf8mb4`, `utf16`, `utf16le`, `utf32`.

  This value can be set at server startup with the `--character_set_server` option.

- `-DDEFAULT_COLLATION=collation_name`

  The server collation. By default, MySQL uses `utf8mb4_0900_ai_ci`. Use the `SHOW COLLATION` statement to determine which collations are available for each character set.

  This value can be set at server startup with the `--collation_server` option.

- `-DDISABLE_PSI_COND=bool`

  Whether to exclude the Performance Schema condition instrumentation. The default is `OFF` (include).

- `-DDISABLE_PSI_FILE=bool`

  Whether to exclude the Performance Schema file instrumentation. The default is `OFF` (include).

- `-DDISABLE_PSI_IDLE=bool`

  Whether to exclude the Performance Schema idle instrumentation. The default is `OFF` (include).

- `-DDISABLE_PSI_MEMORY=bool`

  Whether to exclude the Performance Schema memory instrumentation. The default is `OFF` (include).

- `-DDISABLE_PSI_METADATA=bool`

  Whether to exclude the Performance Schema metadata instrumentation. The default is `OFF` (include).

- `-DDISABLE_PSI_MUTEX=bool`

  Whether to exclude the Performance Schema mutex instrumentation. The default is `OFF` (include).

- `-DDISABLE_PSI_RWLOCK=bool`

  Whether to exclude the Performance Schema rwlock instrumentation. The default is `OFF` (include).

- `-DDISABLE_PSI_SOCKET=bool`

  Whether to exclude the Performance Schema socket instrumentation. The default is `OFF` (include).

- `-DDISABLE_PSI_SP=bool`

  Whether to exclude the Performance Schema stored program instrumentation. The default is `OFF` (include).

- `-DDISABLE_PSI_STAGE=bool`

  Whether to exclude the Performance Schema stage instrumentation. The default is `OFF` (include).

- `-DDISABLE_PSI_STATEMENT=bool`

  Whether to exclude the Performance Schema statement instrumentation. The default is `OFF` (include).

- `-DDISABLE_PSI_STATEMENT_DIGEST=bool`

  Whether to exclude the Performance Schema statement_digest instrumentation. The default is `OFF` (include).

- `-DDISABLE_PSI_TABLE=bool`

  Whether to exclude the Performance Schema table instrumentation. The default is `OFF` (include).

- `-DDISABLE_SHARED=bool`

  Whether to disable building build shared libraries and compile position-dependent code. The default is `OFF` (compile position-independent code).

  This option is unused and was removed in MySQL 8.0.18.

- `-DDISABLE_PSI_PS=bool`

  Exclude the performance schema prepared statements instances instrumentation. The default is `OFF` (include).

- `-DDISABLE_PSI_THREAD=bool`

  Exclude the performance schema thread instrumentation. The default is `OFF` (include).

  Only disable threads when building without any instrumentation, because other instrumentations have a dependency on threads.

- `-DDISABLE_PSI_TRANSACTION=bool`

  Exclude the performance schema transaction instrumentation. The default is `OFF` (include).

- `-DDISABLE_PSI_DATA_LOCK=bool`

  Exclude the performance schema data lock instrumentation. The default is `OFF` (include).

- `-DDISABLE_PSI_ERROR=bool`

  Exclude the performance schema server error instrumentation. The default is `OFF` (include).

- `-DDOWNLOAD_BOOST=bool`

  Whether to download the Boost library. The default is `OFF`.

  See the `WITH_BOOST` option for additional discussion about using Boost.

- `-DDOWNLOAD_BOOST_TIMEOUT=seconds`

  The timeout in seconds for downloading the Boost library. The default is 600 seconds.

  See the `WITH_BOOST` option for additional discussion about using Boost.

- `-DENABLE_DOWNLOADS=bool`

  Whether to download optional files. For example, with this option enabled, `CMake` downloads the Google Test distribution that is used by the test suite to run unit tests, or Ant and JUnit required for building GCS Java wrapper.

  As of MySQL 8.0.26, MySQL source distributions bundle the Google Test source code, used to run Google Test-based unit tests. Consequently, as of that version the `WITH_GMOCK` and `ENABLE_DOWNLOADS` `CMake` options are removed and are ignored if specified.

- `-DENABLE_EXPERIMENTAL_SYSVARS=bool`

  Whether to enable experimental `InnoDB` system variables. Experimental system variables are intended for those engaged in MySQL development, should only be used in a development or test environment, and may be removed without notice in a future MySQL release. For information about experimental system variables, refer to `/storage/innobase/handler/ha_innodb.cc` in the MySQL source tree. Experimental system variables can be identified by searching for “PLUGIN_VAR_EXPERIMENTAL”.

- `-DWITHOUT_SERVER=bool`

  Whether to build without MySQL Server. The default is OFF, which does build the server.

  This is considered an experimental option; it's preferred to build with the server.

- `-DENABLE_GCOV=bool`

  Whether to include gcov support (Linux only).

- `-DENABLE_GPROF=bool`

  Whether to enable `gprof` (optimized Linux builds only).

- `-DENABLED_LOCAL_INFILE=bool`

  This option controls the compiled-in default `LOCAL` capability for the MySQL client library. 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, respectively.

  `ENABLED_LOCAL_INFILE` controls the default for client-side `LOCAL` capability. For the server, the `local_infile` system variable controls server-side `LOCAL` capability. 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, respectively. `local_infile` can also be set at runtime. See Section 6.1.6, “Security Considerations for LOAD DATA LOCAL”.

- `-DENABLED_PROFILING=bool`

  Whether to enable query profiling code (for the `SHOW PROFILE` and `SHOW PROFILES` statements).

- `-DFORCE_UNSUPPORTED_COMPILER=bool`

  By default, `CMake` checks for minimum versions of supported compilers: Visual Studio 2015 (Windows); GCC 4.8 or Clang 3.4 (Linux); Developer Studio 12.5 (Solaris server); Developer Studio 12.4 or GCC 4.8 (Solaris client library); Clang 3.6 (macOS), Clang 3.4 (FreeBSD). To disable this check, use `-DFORCE_UNSUPPORTED_COMPILER=ON`.

- `-DSHOW_SUPPRESSED_COMPILER_WARNINGS=bool`

  Show suppressed compiler warnings, and do so without failing with -Werror. Defaults to OFF.

  This option was added in MySQL 8.0.30.

- `-DFPROFILE_GENERATE=bool`

  Whether to generate profile guided optimization (PGO) data. This option is available for experimenting with PGO with GCC. See the `cmake/fprofile.cmake` file in a MySQL source distribution for information about using `FPROFILE_GENERATE` and `FPROFILE_USE`. These options have been tested with GCC 8 and 9.

  This option was added in MySQL 8.0.19.

- `-DFPROFILE_USE=bool`

  Whether to use profile guided optimization (PGO) data. This option is available for experimenting with PGO with GCC. See the `cmake/fprofile.cmake` file in a MySQL source distribution for information about using `FPROFILE_GENERATE` and `FPROFILE_USE`. These options have been tested with GCC 8 and 9.

  Enabling `FPROFILE_USE` also enables `WITH_LTO`.

  This option was added in MySQL 8.0.19.

- `-DHAVE_PSI_MEMORY_INTERFACE=bool`

  Whether to enable the performance schema memory tracing module for memory allocation functions (`ut::aligned_name` library functions) used in dynamic storage of over-aligned types.

- `-DIGNORE_AIO_CHECK=bool`

If the `-DBUILD_CONFIG=mysql_release` option is given on Linux, the `libaio` library must be linked in by default. If you do not have `libaio` or do not want to install it, you can suppress the check for it by specifying `-DIGNORE_AIO_CHECK=1`.

- `-DMAX_INDEXES=num`

  The maximum number of indexes per table. The default is 64. The maximum is 255. Values smaller than 64 are ignored and the default of 64 is used.

- `-DMYSQL_MAINTAINER_MODE=bool`

  Whether to enable a MySQL maintainer-specific development environment. If enabled, this option causes compiler warnings to become errors.

- `-DWITH_DEVELOPER_ENTITLEMENTS=bool`

  Whether to add the 'get-task-allow' entitlement to all executables to generate a core dump in the event of an unexpected server halt.

  On macOS 11+, core dumps are limited to processes with the com.apple.security.get-task-allow entitlement; which this CMake option enables. The entitlement allows other processes to attach and read/modify the processes memory, and allows *--core-file* to function as expected.

- `-DMUTEX_TYPE=type`

  The mutex type used by `InnoDB`. Options include:

  - `event`: Use event mutexes. This is the default value and the original `InnoDB` mutex implementation.
  - `sys`: Use POSIX mutexes on UNIX systems. Use `CRITICAL_SECTION` objects on Windows, if available.
  - `futex`: Use Linux futexes instead of condition variables to schedule waiting threads.

- `-DMYSQLX_TCP_PORT=port_num`

  The port number on which X Plugin listens for TCP/IP connections. The default is 33060.

  This value can be set at server startup with the `mysqlx_port` system variable.

- `-DMYSQLX_UNIX_ADDR=file_name`

  The Unix socket file path on which the server listens for X Plugin socket connections. This must be an absolute path name. The default is `/tmp/mysqlx.sock`.

  This value can be set at server startup with the `mysqlx_port` system variable.

- `-DMYSQL_PROJECT_NAME=name`

  For Windows or macOS, the project name to incorporate into the project file name.

- `-DMYSQL_TCP_PORT=port_num`

  The port number on which the server listens for TCP/IP connections. The default is 3306.

  This value can be set at server startup with the `--port` option.

- `-DMYSQL_UNIX_ADDR=file_name`

  The Unix socket file path on which the server listens for socket connections. This must be an absolute path name. The default is `/tmp/mysql.sock`.

This value can be set at server startup with the `--socket` option.

- `-DOPTIMIZER_TRACE=bool`

  Whether to support optimizer tracing. See MySQL Internals: Tracing the Optimizer.

- `-DREPRODUCIBLE_BUILD=bool`

  For builds on Linux systems, this option controls whether to take extra care to create a build result independent of build location and time.

  This option was added in MySQL 8.0.11. As of MySQL 8.0.12, it defaults to `ON` for `RelWithDebInfo` builds.

- `-DUSE_LD_GOLD=bool`

  GNU `gold` linker support was removed in v8.0.31; this CMake option was also removed.

  `CMake` causes the build process to link with the GNU `gold` linker if it is available and not explicitly disabled. To disable use of this linker, specify the `-DUSE_LD_GOLD=OFF` option.

- `-DUSE_LD_LLD=bool`

  `CMake` causes the build process to link with the `llvm lld` linker for Clang if it is available and not explicitly disabled. To disable use of this linker, specify the `-DUSE_LD_LLD=OFF` option.

  This option was added in MySQL 8.0.16.

- `-DWIN_DEBUG_NO_INLINE=bool`

  Whether to disable function inlining on Windows. The default is off (inlining enabled).

- `-DWITH_ANT=path_name`

  Set the path to Ant, required when building GCS Java wrapper. Works in a similar way to the existing `WITH_BOOST` CMake option. Set `WITH_ANT` to the path of a directory where the Ant tarball, or an already unpacked archive, is saved. When `WITH_ANT` is not set, or is set with the special value `system`, the build assumes a binary `ant` exists in `$PATH`.

- `-DWITH_ASAN=bool`

  Whether to enable the AddressSanitizer, for compilers that support it. The default is off.

- `-DWITH_ASAN_SCOPE=bool`

  Whether to enable the AddressSanitizer `-fsanitize-address-use-after-scope` Clang flag for use-after-scope detection. The default is off. To use this option, `-DWITH_ASAN` must also be enabled.

- `-DWITH_AUTHENTICATION_CLIENT_PLUGINS=bool`

  This option is enabled automatically if any corresponding server authentication plugins are built. Its value thus depends on other `CMake` options and it should not be set explicitly.

  This option was added in MySQL 8.0.26.

- `-DWITH_AUTHENTICATION_LDAP=bool`

  Whether to report an error if the LDAP authentication plugins cannot be built:

  - If this option is disabled (the default), the LDAP plugins are built if the required header files and libraries are found. If they are not, `CMake` displays a note about it.

  - If this option is enabled, a failure to find the required header file and libraries causes `CMake` to produce an error, preventing the server from being built.

- `-DWITH_AUTHENTICATION_PAM=bool`

  Whether to build the PAM authentication plugin, for source trees that include this plugin. (See Section 6.4.1.5, “PAM Pluggable Authentication”.) If this option is specified and the plugin cannot be compiled, the build fails.

- `-DWITH_AWS_SDK=path_name`

  The location of the Amazon Web Services software development kit.

- `-DWITH_BOOST=path_name`

  The Boost library is required to build MySQL. These `CMake` options enable control over the library source location, and whether to download it automatically:

  - `-DWITH_BOOST=path_name` specifies the Boost library directory location. It is also possible to specify the Boost location by setting the `BOOST_ROOT` or `WITH_BOOST` environment variable.

    `-DWITH_BOOST=system` is also permitted and indicates that the correct version of Boost is installed on the compilation host in the standard location. In this case, the installed version of Boost is used rather than any version included with a MySQL source distribution.

  - `-DDOWNLOAD_BOOST=bool` specifies whether to download the Boost source if it is not present in the specified location. The default is `OFF`.

  - `-DDOWNLOAD_BOOST_TIMEOUT=seconds` the timeout in seconds for downloading the Boost library. The default is 600 seconds.

  For example, if you normally build MySQL placing the object output in the `bld` subdirectory of your MySQL source tree, you can build with Boost like this:

```
mkdir bld
cd bld
cmake .. -DDOWNLOAD_BOOST=ON -DWITH_BOOST=$HOME/my_boost
```

  This causes Boost to be downloaded into the `my_boost` directory under your home directory. If the required Boost version is already there, no download is done. If the required Boost version changes, the newer version is downloaded.

  If Boost is already installed locally and your compiler finds the Boost header files on its own, it may not be necessary to specify the preceding `CMake` options. However, if the version of Boost required by MySQL changes and the locally installed version has not been upgraded, you may have build problems. Using the `CMake` options should give you a successful build.

  With the above settings that allow Boost download into a specified location, when the required Boost version changes, you need to remove the `bld` folder, recreate it, and perform the `cmake` step again. Otherwise, the new Boost version might not get downloaded, and compilation might fail.

- `-DWITH_CLIENT_PROTOCOL_TRACING=bool`

  Whether to build the client-side protocol tracing framework into the client library. By default, this option is enabled.

  For information about writing protocol trace client plugins, see Writing Protocol Trace Plugins.

  See also the `WITH_TEST_TRACE_PLUGIN` option.

- `-DWITH_CURL=curl_type`

  The location of the `curl` library. *`curl_type`* can be `system` (use the system `curl` library) or a path name to the `curl` library.

- `-DWITH_DEBUG=bool`

  Whether to include debugging support.

  Configuring MySQL with debugging support enables you to use the `--debug="d,parser_debug"` option when you start the server. This causes the Bison parser that is used to process SQL statements to dump a parser trace to the server's standard error output. Typically, this output is written to the error log.

  Sync debug checking for the `InnoDB` storage engine is defined under `UNIV_DEBUG` and is available when debugging support is compiled in using the `WITH_DEBUG` option. When debugging support is compiled in, the `innodb_sync_debug` configuration option can be used to enable or disable `InnoDB` sync debug checking.

  Enabling `WITH_DEBUG` also enables Debug Sync. This facility is used for testing and debugging. When compiled in, Debug Sync is disabled by default at runtime. To enable it, start `mysqld` with the `--debug-sync-timeout=N` option, where *`N`* is a timeout value greater than 0. (The default value is 0, which disables Debug Sync.) *`N`* becomes the default timeout for individual synchronization points.

  Sync debug checking for the `InnoDB` storage engine is available when debugging support is compiled in using the `WITH_DEBUG` option.

  For a description of the Debug Sync facility and how to use synchronization points, see MySQL Internals: Test Synchronization.

- `-DWITH_DEFAULT_FEATURE_SET=bool`

  Whether to use the flags from `cmake/build_configurations/feature_set.cmake`. This option was removed in MySQL 8.0.22.

- `-DWITH_EDITLINE=value`

  Which `libedit`/`editline` library to use. The permitted values are `bundled` (the default) and `system`.

- `-DWITH_FIDO=fido_type`

  The `authentication_fido` authentication plugin is implemented using a FIDO library (see Section 6.4.1.11, “FIDO Pluggable Authentication”). The `WITH_FIDO` option indicates the source of FIDO support:

  - `bundled`: Use the FIDO library bundled with the distribution. This is the default.

    As of MySQL 8.0.30, MySQL includes `fido2` version 1.8.0. (Prior releases used `fido2` 1.5.0).

  - `system`: Use the system FIDO library.

  This option was added in MySQL 8.0.27.

- `-DWITH_GMOCK=path_name`

  The path to the googlemock distribution, for use with Google Test-based unit tests. The option value is the path to the distribution Zip file. Alternatively, set the `WITH_GMOCK` environment variable to

the path name. It is also possible to use `-DENABLE_DOWNLOADS=1`, so that `CMake` downloads the distribution from GitHub.

If you build MySQL without the Google Test-based unit tests (by configuring without `WITH_GMOCK`), `CMake` displays a message indicating how to download it.

As of MySQL 8.0.26, MySQL source distributions bundle the Google Test source code, used to run Google Test-based unit tests. Consequently, as of that version the `WITH_GMOCK` and `ENABLE_DOWNLOADS` `CMake` options are removed and are ignored if specified.

- `-DWITH_ICU={`*`icu_type`*`|`*`path_name`*`}`

  MySQL uses International Components for Unicode (ICU) to support regular expression operations. The `WITH_ICU` option indicates the type of ICU support to include or the path name to the ICU installation to use.

  - *`icu_type`* can be one of the following values:

    - `bundled`: Use the ICU library bundled with the distribution. This is the default, and is the only supported option for Windows.

    - `system`: Use the system ICU library.

  - *`path_name`* is the path name to the ICU installation to use. This can be preferable to using the *`icu_type`* value of `system` because it can prevent CMake from detecting and using an older or incorrect ICU version installed on the system. (Another permitted way to do the same thing is to set `WITH_ICU` to `system` and set the `CMAKE_PREFIX_PATH` option to *`path_name`*.)

- `-DWITH_INNODB_EXTRA_DEBUG=`*`bool`*

  Whether to include extra InnoDB debugging support.

  Enabling `WITH_INNODB_EXTRA_DEBUG` turns on extra InnoDB debug checks. This option can only be enabled when `WITH_DEBUG` is enabled.

- `-DWITH_INNODB_MEMCACHED=`*`bool`*

  Whether to generate memcached shared libraries (`libmemcached.so` and `innodb_engine.so`).

- `-DWITH_JEMALLOC=`*`bool`*

  Whether to link with `-ljemalloc`. If enabled, built-in `malloc()`, `calloc()`, `realloc()`, and `free()` routines are disabled. The default is `OFF`.

  `WITH_JEMALLOC` and `WITH_TCMALLOC` are mutually exclusive.

  This option was added in MySQL 8.0.16.

- `-DWITH_WIN_JEMALLOC=`*`string`*

  On Windows, pass in a path to a directory containing `jemalloc.dll` to enable jemalloc functionality. The build system copies `jemalloc.dll` to the same directory as `mysqld.exe` and/or `mysqld-debug.exe` and utilizes it for memory management operations. Standard memory functions are used if `jemalloc.dll` is not found or does not export the required functions. An INFORMATION level log message records whether or not jemalloc is found and used.

  This option is enabled for official MySQL binaries for Windows.

  This option was added in MySQL 8.0.29.

- `-DWITH_KEYRING_TEST=bool`

  Whether to build the test program that accompanies the `keyring_file` plugin. The default is `OFF`. Test file source code is located in the `plugin/keyring/keyring-test` directory.

- `-DWITH_LIBEVENT=string`

  Which `libevent` library to use. Permitted values are `bundled` (default) and `system`. Prior to MySQL 8.0.21, if you specify `system`, the system `libevent` library is used if present, and an error occurs otherwise. In MySQL 8.0.21 and later, if `system` is specified and no system `libevent` library can be found, an error occurs regardless, and the bundled `libevent` is not used.

  The `libevent` library is required by `InnoDB` memcached, X Plugin, and MySQL Router.

- `-DWITH_LIBWRAP=bool`

  Whether to include `libwrap` (TCP wrappers) support.

- `-DWITH_LOCK_ORDER=bool`

  Whether to enable LOCK_ORDER tooling. By default, this option is disabled and server builds contain no tooling. If tooling is enabled, the LOCK_ORDER tool is available and can be used as described in Section 5.9.3, “The LOCK_ORDER Tool”.

  Note

  With the `WITH_LOCK_ORDER` option enabled, MySQL builds require the `flex` program.

  This option was added in MySQL 8.0.17.

- `-DWITH_LSAN=bool`

  Whether to run LeakSanitizer, without AddressSanitizer. The default is `OFF`.

  This option was added in MySQL 8.0.16.

- `-DWITH_LTO=bool`

  Whether to enable the link-time optimizer, if the compiler supports it. The default is `OFF` unless `FPROFILE_USE` is enabled.

  This option was added in MySQL 8.0.13.

- `-DWITH_LZ4=lz4_type`

  The `WITH_LZ4` option indicates the source of `zlib` support:

  - `bundled`: Use the `lz4` library bundled with the distribution. This is the default.
  - `system`: Use the system `lz4` library. If `WITH_LZ4` is set to this value, the `lz4_decompress` utility is not built. In this case, the system `lz4` command can be used instead.

- `-DWITH_LZMA=lzma_type`

  The type of LZMA library support to include. *`lzma_type`* can be one of the following values:

  - `bundled`: Use the LZMA library bundled with the distribution. This is the default.
  - `system`: Use the system LZMA library.

  This option was removed in MySQL 8.0.16.

- `-DWITH_MECAB={disabled|system|path_name}`

  Use this option to compile the MeCab parser. If you have installed MeCab to its default installation directory, set `-DWITH_MECAB=system`. The `system` option applies to MeCab installations performed from source or from binaries using a native package management utility. If you installed MeCab to a custom installation directory, specify the path to the MeCab installation. For example, `-DWITH_MECAB=/opt/mecab`. If the `system` option does not work, specifying the MeCab installation path should work in all cases.

  For related information, see Section 12.10.9, “MeCab Full-Text Parser Plugin”.

- `-DWITH_MSAN=bool`

  Whether to enable MemorySanitizer, for compilers that support it. The default is off.

  For this option to have an effect if enabled, all libraries linked to MySQL must also have been compiled with the option enabled.

- `-DWITH_MSCRT_DEBUG=bool`

  Whether to enable Visual Studio CRT memory leak tracing. The default is `OFF`.

- `-DMSVC_CPPCHECK=bool`

  Whether to enable MSVC code analysis. The default is `OFF`.

- `-DWITH_MYSQLX=bool`

  Whether to build with support for X Plugin. Default `ON`. See Chapter 20, *Using MySQL as a Document Store*.

- `-DWITH_NUMA=bool`

  Explicitly set the NUMA memory allocation policy. `CMake` sets the default `WITH_NUMA` value based on whether the current platform has `NUMA` support. For platforms without NUMA support, `CMake` behaves as follows:

  - With no NUMA option (the normal case), `CMake` continues normally, producing only this warning: NUMA library missing or required version not available
  - With `-DWITH_NUMA=ON`, `CMake` aborts with this error: NUMA library missing or required version not available

- `-DWITH_PACKAGE_FLAGS=bool`

  For flags typically used for RPM and Debian packages, whether to add them to standalone builds on those platforms. The default is `ON` for nondebug builds.

  This option was added in MySQL 8.0.26.

- `-DWITH_PROTOBUF=protobuf_type`

  Which Protocol Buffers package to use. *`protobuf_type`* can be one of the following values:

  - `bundled`: Use the package bundled with the distribution. This is the default. Optionally use `INSTALL_PRIV_LIBDIR` to modify the dynamic Protobuf library directory.
  - `system`: Use the package installed on the system.

  Other values are ignored, with a fallback to `bundled`.

- `-DWITH_RAPID=bool`

  Whether to build the rapid development cycle plugins. When enabled, a `rapid` directory is created in the build tree containing these plugins. When disabled, no `rapid` directory is created in the build tree. The default is `ON`, unless the `rapid` directory is removed from the source tree, in which case the default becomes `OFF`.

- `-DWITH_RAPIDJSON=rapidjson_type`

  The type of RapidJSON library support to include. *`rapidjson_type`* can be one of the following values:

  - `bundled`: Use the RapidJSON library bundled with the distribution. This is the default.
  - `system`: Use the system RapidJSON library. Version 1.1.0 or higher is required.

  This option was added in MySQL 8.0.13.

- `-DWITH_RE2=re2_type`

  The type of RE2 library support to include. *`re2_type`* can be one of the following values:

  - `bundled`: Use the RE2 library bundled with the distribution. This is the default.
  - `system`: Use the system RE2 library.

  As of MySQL 8.0.18, MySQL no longer uses the RE2 library and this option was removed.

- `-DWITH_ROUTER=bool`

  Whether to build MySQL Router. The default is `ON`.

  This option was added in MySQL 8.0.16.

- `-DWITH_SSL={ssl_type|path_name}`

  For support of encrypted connections, entropy for random number generation, and other encryption-related operations, MySQL must be built using an SSL library. This option specifies which SSL library to use.

  - *`ssl_type`* can be one of the following values:

    - `system`: Use the system OpenSSL library. This is the default.

      On macOS and Windows, using `system` configures MySQL to build as if CMake was invoked with *`path_name`* points to a manually installed OpenSSL library. This is because they do not have system SSL libraries. On macOS, *brew install openssl* installs to `/usr/local/opt/openssl` so that `system` can find it. On Windows, it checks *%ProgramFiles%/OpenSSL*, *%ProgramFiles%/OpenSSL-Win32*, *%ProgramFiles%/OpenSSL-Win64*, *C:/OpenSSL*, *C:/OpenSSL-Win32*, and *C:/OpenSSL-Win64*.

    - `yes`: This is a synonym for `system`.
    - `openssl[\d]`: Uses an alternate OpenSSL system package such as *openssl11* on EL7 or *openssl3* on EL8. Support was added in v8.0.30.

      Authentication plugins, such as LDAP and Kerberos, are disabled as they do not support these alternative versions of OpenSSL.

  - *`path_name`* is the path name to the OpenSSL installation to use. This can be preferable to using the *`ssl_type`* value of `system` because it can prevent CMake from detecting and using an older

or incorrect OpenSSL version installed on the system. (Another permitted way to do the same thing is to set `WITH_SSL` to `system` and set the `CMAKE_PREFIX_PATH` option to *`path_name`*.)

For additional information about configuring the SSL library, see Section 2.8.6, “Configuring SSL Library Support”.

- `-DWITH_SYSTEMD=`*`bool`*

  Whether to enable installation of systemd support files. By default, this option is disabled. When enabled, systemd support files are installed, and scripts such as `mysqld_safe` and the System V initialization script are not installed. On platforms where systemd is not available, enabling `WITH_SYSTEMD` results in an error from `CMake`.

  For more information about using systemd, see Section 2.5.9, “Managing MySQL Server with systemd”. That section also includes information about specifying options previously specified in `[mysqld_safe]` option groups. Because `mysqld_safe` is not installed when systemd is used, such options must be specified another way.

- `-DWITH_SYSTEM_LIBS=`*`bool`*

  This option serves as an “umbrella” option to set the `system` value of any of the following `CMake` options that are not set explicitly: `WITH_CURL`, `WITH_EDITLINE`, `WITH_FIDO`, `WITH_ICU`, `WITH_LIBEVENT`, `WITH_LZ4`, `WITH_LZMA`, `WITH_PROTOBUF`, `WITH_RE2`, `WITH_SSL`, `WITH_ZSTD`.

  `WITH_ZLIB` was included here before v8.0.30.

- `-DWITH_SYSTEMD_DEBUG=`*`bool`*

  Whether to produce additional systemd debugging information, for platforms on which systemd is used to run MySQL. The default is `OFF`.

  This option was added in MySQL 8.0.22.

- `-DWITH_TCMALLOC=`*`bool`*

  Whether to link with `-ltcmalloc`. If enabled, built-in `malloc()`, `calloc()`, `realloc()`, and `free()` routines are disabled. The default is `OFF`.

  `WITH_TCMALLOC` and `WITH_JEMALLOC` are mutually exclusive.

  This option was added in MySQL 8.0.22.

- `-DWITH_TEST_TRACE_PLUGIN=`*`bool`*

  Whether to build the test protocol trace client plugin (see Using the Test Protocol Trace Plugin). By default, this option is disabled. Enabling this option has no effect unless the `WITH_CLIENT_PROTOCOL_TRACING` option is enabled. If MySQL is configured with both options enabled, the `libmysqlclient` client library is built with the test protocol trace plugin built in, and all the standard MySQL clients load the plugin. However, even when the test plugin is enabled, it has no effect by default. Control over the plugin is afforded using environment variables; see Using the Test Protocol Trace Plugin.

  Note

  Do *not* enable the `WITH_TEST_TRACE_PLUGIN` option if you want to use your own protocol trace plugins because only one such plugin can be loaded at a time and an error occurs for attempts to load a second one. If you have already built MySQL with the test protocol trace plugin enabled to see how it works, you must rebuild MySQL without it before you can use your own plugins.

  For information about writing trace plugins, see Writing Protocol Trace Plugins.

- `-DWITH_TSAN=bool`

  Whether to enable the ThreadSanitizer, for compilers that support it. The default is off.

- `-DWITH_UBSAN=bool`

  Whether to enable the Undefined Behavior Sanitizer, for compilers that support it. The default is off.

- `-DWITH_UNIT_TESTS={ON|OFF}`

  If enabled, compile MySQL with unit tests. The default is ON unless the server is not being compiled.

- `-DWITH_UNIXODBC=1`

  Enables unixODBC support, for Connector/ODBC.

- `-DWITH_VALGRIND=bool`

  Whether to compile in the Valgrind header files, which exposes the Valgrind API to MySQL code. The default is `OFF`.

  To generate a Valgrind-aware debug build, `-DWITH_VALGRIND=1` normally is combined with `-DWITH_DEBUG=1`. See Building Debug Configurations.

- `-DWITH_ZLIB=zlib_type`

  Some features require that the server be built with compression library support, such as the `COMPRESS()` and `UNCOMPRESS()` functions, and compression of the client/server protocol. The `WITH_ZLIB` option indicates the source of `zlib` support:

  The minimum `zlib` version supported is 1.2.12 as of MySQL 8.0.30.

  - `bundled`: Use the `zlib` library bundled with the distribution. This is the default.
  - `system`: Use the system `zlib` library. If `WITH_ZLIB` is set to this value, the `zlib_decompress` utility is not built. In this case, the system `openssl zlib` command can be used instead.

- `-DWITH_ZSTD=zstd_type`

  Connection compression using the `zstd` algorithm (see Section 4.2.8, “Connection Compression Control”) requires that the server be built with `zstd` library support. The `WITH_ZSTD` option indicates the source of `zstd` support:

  - `bundled`: Use the `zstd` library bundled with the distribution. This is the default.
  - `system`: Use the system `zstd` library.

  This option was added in MySQL 8.0.18.

Compiler Flags

- `-DCMAKE_C_FLAGS="flags"`

  Flags for the C Compiler.

- `-DCMAKE_CXX_FLAGS="flags"`

  Flags for the C++ Compiler.

- `-DWITH_DEFAULT_COMPILER_OPTIONS=bool`

  Whether to use the flags from `cmake/build_configurations/compiler_options.cmake`.

Note

All optimization flags were carefully chosen and tested by the MySQL build team. Overriding them can lead to unexpected results and is done at your own risk.

To specify your own C and C++ compiler flags, for flags that do not affect optimization, use the `CMAKE_C_FLAGS` and `CMAKE_CXX_FLAGS` CMake options.

When providing your own compiler flags, you might want to specify `CMAKE_BUILD_TYPE` as well.

For example, to create a 32-bit release build on a 64-bit Linux machine, do this:

```
mkdir bld
cd bld
cmake .. -DCMAKE_C_FLAGS=-m32 \
  -DCMAKE_CXX_FLAGS=-m32 \
  -DCMAKE_BUILD_TYPE=RelWithDebInfo
```

If you set flags that affect optimization (`-O`*`number`*), you must set the `CMAKE_C_FLAGS_`*`build_type`* and/or `CMAKE_CXX_FLAGS_`*`build_type`* options, where *`build_type`* corresponds to the `CMAKE_BUILD_TYPE` value. To specify a different optimization for the default build type (`RelWithDebInfo`) set the `CMAKE_C_FLAGS_RELWITHDEBINFO` and `CMAKE_CXX_FLAGS_RELWITHDEBINFO` options. For example, to compile on Linux with `-O3` and with debug symbols, do this:

```
cmake .. -DCMAKE_C_FLAGS_RELWITHDEBINFO="-O3 -g" \
  -DCMAKE_CXX_FLAGS_RELWITHDEBINFO="-O3 -g"
```

CMake Options for Compiling NDB Cluster

The following options are for use when building MySQL 8.0 sources with NDB Cluster support.

- `-DMEMCACHED_HOME=`*`dir_name`*

  `NDB` support for memcached was removed in NDB 8.0.23; thus, this option is no longer supported for building `NDB` in this or later versions.

- `-DNDB_UTILS_LINK_DYNAMIC={ON|OFF}`

  Controls whether NDB utilities such as `ndb_drop_table` are linked with `ndbclient` statically (`OFF`) or dynamically (`ON`); `OFF` (static linking) is the default. Normally static linking is used when building these to avoid problems with `LD_LIBRARY_PATH`, or when multiple versions of `ndbclient` are installed. This option is intended for creating Docker images and possibly other cases in which the target environment is subject to precise control and it is desirable to reduce image size.

  Added in NDB 8.0.22.

- `-DWITH_BUNDLED_LIBEVENT={ON|OFF}`

  `NDB` support for memcached was removed in NDB 8.0.23; thus, this option is no longer supported for building `NDB` in this or later versions.

- `-DWITH_BUNDLED_MEMCACHED={ON|OFF}`

  `NDB` support for memcached was removed in NDB 8.0.23; thus, this option is no longer supported for building `NDB` in this or later versions.

- `-DWITH_CLASSPATH=`*`path`*

  Sets the classpath for building NDB Cluster Connector for Java. The default is empty. This option is ignored if `-DWITH_NDB_JAVA=OFF` is used.

- `-DWITH_ERROR_INSERT={ON|OFF}`

  Enables error injection in the `NDB` kernel. For testing only; not intended for use in building production binaries. The default is `OFF`.

- `-DWITH_NDB={ON|OFF}`

  Build MySQL NDB Cluster; build the NDB plugin and all NDB programs.

  Added in NDB 8.0.31.

- `-DWITH_NDBAPI_EXAMPLES={ON|OFF}`

  Build API example programs in `storage/ndb/ndbapi-examples/`.

- `-DWITH_NDBCLUSTER_STORAGE_ENGINE={ON|OFF}`

  NDB 8.0.30 and earlier: For internal use only; may not always work as expected. To build with `NDB` support, use `WITH_NDBCLUSTER` instead.

  NDB 8.0.31 and later: Controls (only) whether the `ndbcluster` plugin is included in the build; `WITH_NDB` enables this option automatically, so it is recommended that you use that `WITH_NDB` instead.

- `-DWITH_NDBCLUSTER={ON|OFF}`

  Build and link in support for the `NDB` storage engine in `mysqld`.

  This option is deprecated in NDB 8.0.31, and subject to eventual removal; use `WITH_NDB` instead.

- `-DWITH_NDBMTD={ON|OFF}`

  Build the multithreaded data node executable `ndbmtd`. The default is `ON`.

- `-DWITH_NDB_DEBUG={ON|OFF}`

  Enable building the debug versions of the NDB Cluster binaries. OFF by default.

- `-DWITH_NDB_JAVA={ON|OFF}`

  Enable building NDB Cluster with Java support, including `ClusterJ`.

  This option is ON by default. If you do not wish to compile NDB Cluster with Java support, you must disable it explicitly by specifying `-DWITH_NDB_JAVA=OFF` when running `CMake`. Otherwise, if Java cannot be found, configuration of the build fails.

- `-DWITH_NDB_PORT=`*`port`*

  Causes the NDB Cluster management server (`ndb_mgmd`) that is built to use this *`port`* by default. If this option is unset, the resulting management server tries to use port 1186 by default.

- `-DWITH_NDB_TEST={ON|OFF}`

  If enabled, include a set of NDB API test programs. The default is OFF.

- `-DWITH_PLUGIN_NDBCLUSTER={ON|OFF}`

  For internal use only; may not always work as expected. This option is removed in NDB 8.0.31; use `WITH_NDB` instead to build MySQL Cluster. (*NDB 8.0.30 and earlier*: Use `WITH_NDBCLUSTER`.)

2.8.8 Dealing with Problems Compiling MySQL

The solution to many problems involves reconfiguring. If you do reconfigure, take note of the following:

- If `CMake` is run after it has previously been run, it may use information that was gathered during its previous invocation. This information is stored in `CMakeCache.txt`. When `CMake` starts, it looks for that file and reads its contents if it exists, on the assumption that the information is still correct. That assumption is invalid when you reconfigure.
- Each time you run `CMake`, you must run `make` again to recompile. However, you may want to remove old object files from previous builds first because they were compiled using different configuration options.

To prevent old object files or configuration information from being used, run the following commands before re-running `CMake`:

On Unix:

```
$> make clean
$> rm CMakeCache.txt
```

On Windows:

```
$> devenv MySQL.sln /clean
$> del CMakeCache.txt
```

If you build outside of the source tree, remove and recreate your build directory before re-running `CMake`. For instructions on building outside of the source tree, see How to Build MySQL Server with CMake.

On some systems, warnings may occur due to differences in system include files. The following list describes other problems that have been found to occur most often when compiling MySQL:

- To define which C and C++ compilers to use, you can define the `CC` and `CXX` environment variables. For example:

  ```
  $> CC=gcc
  $> CXX=g++
  $> export CC CXX
  ```

  To specify your own C and C++ compiler flags, use the `CMAKE_C_FLAGS` and `CMAKE_CXX_FLAGS` CMake options. See Compiler Flags.

  To see what flags you might need to specify, invoke `mysql_config` with the `--cflags` and `--cxxflags` options.

- To see what commands are executed during the compile stage, after using `CMake` to configure MySQL, run `make VERBOSE=1` rather than just `make`.
- If compilation fails, check whether the `MYSQL_MAINTAINER_MODE` option is enabled. This mode causes compiler warnings to become errors, so disabling it may enable compilation to proceed.
- If your compile fails with errors such as any of the following, you must upgrade your version of `make` to GNU `make`:

  ```
  make: Fatal error in reader: Makefile, line 18:
  Badly formed macro assignment
  ```

  Or:

  ```
  make: file `Makefile' line 18: Must be a separator (:
  ```

  Or:

  ```
  pthread.h: No such file or directory
  ```

  Solaris and FreeBSD are known to have troublesome `make` programs.

  GNU `make` 3.75 is known to work.

- The `sql_yacc.cc` file is generated from `sql_yacc.yy`. Normally, the build process does not need to create `sql_yacc.cc` because MySQL comes with a pregenerated copy. However, if you do need to re-create it, you might encounter this error:

  ```
  "sql_yacc.yy", line xxx fatal: default action causes potential...
  ```

  This is a sign that your version of `yacc` is deficient. You probably need to install a recent version of `bison` (the GNU version of `yacc`) and use that instead.

  Versions of `bison` older than 1.75 may report this error:

  ```
  sql_yacc.yy:#####: fatal error: maximum table size (32767) exceeded
  ```

  The maximum table size is not actually exceeded; the error is caused by bugs in older versions of `bison`.

For information about acquiring or updating tools, see the system requirements in Section 2.8, “Installing MySQL from Source”.

2.8.9 MySQL Configuration and Third-Party Tools

Third-party tools that need to determine the MySQL version from the MySQL source can read the `VERSION` file in the top-level source directory. The file lists the pieces of the version separately. For example, if the version is MySQL 8.0.4-rc, the file looks like this:

```
MYSQL_VERSION_MAJOR=8
MYSQL_VERSION_MINOR=0
MYSQL_VERSION_PATCH=4
MYSQL_VERSION_EXTRA=-rc
```

If the source is not for a General Availability (GA) release, the `MYSQL_VERSION_EXTRA` value is nonempty. In the example just shown, the value corresponds to “Release Candidate”.

To construct a five-digit number from the version components, use this formula:

```
MYSQL_VERSION_MAJOR*10000 + MYSQL_VERSION_MINOR*100 + MYSQL_VERSION_PATCH
```

2.8.10 Generating MySQL Doxygen Documentation Content

The MySQL source code contains internal documentation written using Doxygen. The generated Doxygen content is available at https://dev.mysql.com/doc/index-other.html. It is also possible to generate this content locally from a MySQL source distribution using the following procedure:

1. Install `doxygen` 1.9.2 or higher. Distributions are available here at http://www.doxygen.nl/.

   After installing `doxygen`, verify the version number:

   ```
   $> doxygen --version
   1.9.2
   ```

2. Install PlantUML.

   When you install PlantUML on Windows (tested on Windows 10), you must run it at least once as administrator so it creates the registry keys. Open an administrator console and run this command:

   ```
   $> java -jar path-to-plantuml.jar
   ```

   The command should open a GUI window and return no errors on the console.

3. Set the `PLANTUML_JAR_PATH` environment to the location where you installed PlantUML. For example:

   ```
   $> export PLANTUML_JAR_PATH=path-to-plantuml.jar
   ```

4. Install the Graphviz `dot` command.

   After installing Graphviz, verify `dot` availability. For example:

   ```
   $> which dot
   /usr/bin/dot

   $> dot -V
   dot - graphviz version 2.28.0 (20130928.0220)
   ```

5. Change location to the top-level directory of your MySQL source distribution and do the following:

   First, execute `cmake`:

   ```
   $> cd your-mysql-source-directory
   $> mkdir bld
   $> cd bld
   $> cmake ..
   ```

   Next, generate the `doxygen` documentation:

   ```
   $> make doxygen
   ```

   Inspect the error log. It is available in the `doxyerror.log` file in the top-level directory. Assuming that the build executed successfully, view the generated output using a browser. For example:

   ```
   $> firefox doxygen/html/index.html
   ```

2.9 Postinstallation Setup and Testing

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 system, and installation from a ZIP Archive package on Windows. For instructions, see Section 2.9.1, “Initializing the Data Directory”.
- Start the server and make sure that it can be accessed. For instructions, see Section 2.9.2, “Starting the Server”, and Section 2.9.3, “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 2.9.4, “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 2.9.5, “Starting and Stopping MySQL Automatically”.
- Optionally, populate time zone tables to enable recognition of named time zones. For instructions, see Section 5.1.15, “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 Section 6.2, “Access Control and Account Management”.

2.9.1 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 Section 2.9, “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 2.9.3, “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):

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

   Within this directory you can 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:

   ```
   mkdir mysql-files
   ```

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

   ```
   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:

   ```
   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 Section 2.10, “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:

   ```
   bin/mysql_ssl_rsa_setup
   ```

   For more information, see Section 4.4.3, “mysql_ssl_rsa_setup — Create SSL/RSA Files”.

5. In the absence of any option files, the server starts with its default settings. (See Section 5.1.2, “Server Configuration Defaults”.) To explicitly specify options that the MySQL server should use at startup, put them in an option file such as `/etc/my.cnf` or `/etc/mysql/my.cnf`. (See Section 4.2.2.2, “Using Option Files”.) For example, you can use an option file to set the `secure_file_priv` system variable.

6. To arrange for MySQL to start without manual intervention at system boot time, see Section 2.9.5, “Starting and Stopping MySQL Automatically”.

7. Data directory initialization creates time zone tables in the `mysql` schema but does not populate them. To do so, use the instructions in Section 5.1.15, “MySQL Server Time Zone Support”.

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 must choose a new one.
- With `--initialize-insecure`, no `root` password is generated. This is insecure; it is assumed that you intend to assign a password to the account in a 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 Section 5.4.2, “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 `mysqld` 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:

```
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):

```
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:

```
[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):

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

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

```
[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):

```
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 Section 4.4.2, “mysql_secure_installation — Improve MySQL Installation Security”, and Section 4.4.3, “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:

     ```
     [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 Section 5.3, “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 6.2.9, “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 Section 13.8.3, “HELP Statement”). The server does not populate the time zone tables. To do so manually, see Section 5.1.15, “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 2.9.2, “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 2.9.4, “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.

2.9.2 Starting the Server

This section describes how start the server on Unix and Unix-like systems. (For Windows, see Section 2.3.4.5, “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 2.9.3, “Testing the Server”.

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

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

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 Section 2.5.9, “Managing MySQL Server with systemd”.

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

```
$> 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 6.1.5, “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 2.9.2.1, “Troubleshooting Problems Starting the MySQL Server”. For more information about `mysqld_safe`, see Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”. For more information about systemd support, see Section 2.5.9, “Managing MySQL Server with systemd”.

2.9.2.1 Troubleshooting Problems Starting the MySQL Server

This section provides troubleshooting suggestions for problems starting the server. For additional suggestions for Windows systems, see Section 2.3.5, “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:

```
$> 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 Section 15.8, "InnoDB Configuration" for guidelines and Section 15.14, "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.

- 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 does 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`:

  ```
  $> ./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:

  ```
  $> mysqladmin variables
  ```

  Or:

  ```
  $> 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:

```
$> 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:

```
$> chown -R mysql /usr/local/mysql/var
$> 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.

- 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 Section 5.8, “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 Section 5.9.4, “The DBUG Package”.

2.9.3 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 Section 4.2.9, “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:

```
$> bin/mysqladmin version
$> 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:

```
$> 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:

```
$> bin/mysqladmin version
mysqladmin  Ver 14.12 Distrib 8.0.32, for pc-linux-gnu on i686
...

Server version          8.0.32
Protocol version        10
Connection              Localhost via UNIX socket
UNIX socket             /var/lib/mysql/mysql.sock
Uptime:                 14 days 5 hours 5 min 21 sec

Threads: 1  Questions: 366  Slow queries: 0
Opens: 0  Flush tables: 1  Open tables: 19
Queries per second avg: 0.000
```

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):

```
$> 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:

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

If `mysqld_safe` fails, see Section 2.9.2.1, “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:

```
$> bin/mysqlshow
+--------------------+
|     Databases      |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
```

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:

```
$> bin/mysqlshow mysql
Database: mysql
+---------------------------+
|          Tables           |
+---------------------------+
```

```
| columns_priv              |
| component                 |
| db                        |
| default_roles             |
| engine_cost               |
| func                      |
| general_log               |
| global_grants             |
| gtid_executed             |
| help_category             |
| help_keyword              |
| help_relation             |
| help_topic                |
| innodb_index_stats        |
| innodb_table_stats        |
| ndb_binlog_index          |
| password_history          |
| plugin                    |
| procs_priv                |
| proxies_priv              |
| role_edges                |
| server_cost               |
| servers                   |
| slave_master_info         |
| slave_relay_log_info      |
| slave_worker_info         |
| slow_log                  |
| tables_priv               |
| time_zone                 |
| time_zone_leap_second     |
| time_zone_name            |
| time_zone_transition      |
| time_zone_transition_type |
| user                      |
+---------------------------+
```

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

```
$> bin/mysql -e "SELECT User, Host, plugin FROM mysql.user" mysql
+------+-----------+-----------------------+
| User | Host      | plugin                |
+------+-----------+-----------------------+
| root | localhost | caching_sha2_password |
+------+-----------+-----------------------+
```

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 2.9.4, “Securing the Initial MySQL Account”.

For more information about `mysql`, `mysqladmin`, and `mysqlshow`, see Section 4.5.1, “mysql — The MySQL Command-Line Client”, Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”, and Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”.

2.9.4 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 2.9.1, “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 Section 2.3.3, “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 Chapter 31, *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.
- 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 2.9.1, “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 6.2.19, “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.

Start the server if it is not running. For instructions, see Section 2.9.2, “Starting the Server”.

The initial `root` account may or may not have a password. Choose whichever of the following procedures applies:

- If the `root` account exists with an initial random password that has been expired, connect to the server as `root` using that password, then choose a new password. This is the case if the data directory was initialized using `mysqld --initialize`, either manually or using an installer that does not give you the option of specifying a password during the install operation. Because the password exists, you must use it to connect to the server. But because the password is expired, you cannot use the account for any purpose other than to choose a new password, until you do choose one.

  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:

     ```
     $> 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:

     ```
     $> 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:

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

To shut down the server with `mysqladmin`, use this command:

```
$> mysqladmin -u root -p shutdown
Enter password: (enter root password here)
```

Note

For additional information about setting passwords, see Section 6.2.14, “Assigning Account Passwords”. If you forget your `root` password after setting it, see Section B.3.3.2, “How to Reset the Root Password”.

To set up additional accounts, see Section 6.2.8, “Adding Accounts, Assigning Privileges, and Dropping Accounts”.

2.9.5 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 Section 2.3.4.8, “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 Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”.
- On Linux systems that support systemd, you can use it to control the server. See Section 2.5.9, “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 Section 4.3.3, “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 Section 2.4.3, “Installing and Using the MySQL Launch Daemon”. A MySQL Preference Pane also provides control for starting and stopping MySQL through the System Preferences. See Section 2.4.4, “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 2.15 MySQL Startup Scripts and Supported Server Option Groups

Script	Option Groups
`mysqld`	`[mysqld]`, `[server]`, `[mysqld-major_version]`
`mysqld_safe`	`[mysqld]`, `[server]`, `[mysqld_safe]`
`mysql.server`	`[mysqld]`, `[mysql.server]`, `[server]`

`[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 Section 4.2.2.2, “Using Option Files”.

2.10 Upgrading MySQL

This section describes the steps to upgrade a MySQL installation.

Upgrading is a common procedure, as you pick up bug fixes within the same MySQL release series or significant features between major MySQL releases. You perform this procedure first on some test systems to make sure everything works smoothly, and then on the production systems.

Note

In the following discussion, MySQL commands that must be run using a MySQL account with administrative privileges include `-u root` on the command line to specify the MySQL `root` user. Commands that require a password for `root` also include a `-p` option. Because `-p` is followed by no option value, such commands prompt for the password. Type the password when prompted and press Enter.

SQL statements can be executed using the `mysql` command-line client (connect as `root` to ensure that you have the necessary privileges).

2.10.1 Before You Begin

Review the information in this section before upgrading. Perform any recommended actions.

- Understand what may occur during an upgrade. See Section 2.10.3, “What the MySQL Upgrade Process Upgrades”.
- Protect your data by creating a backup. The backup should include the `mysql` system database, which contains the MySQL data dictionary tables and system tables. See Section 7.2, “Database Backup Methods”.

Important

Downgrade from MySQL 8.0 to MySQL 5.7, or from a MySQL 8.0 release to a previous MySQL 8.0 release, is not supported. The only supported alternative

is to restore a backup taken *before* upgrading. It is therefore imperative that you back up your data before starting the upgrade process.

- Review Section 2.10.2, “Upgrade Paths” to ensure that your intended upgrade path is supported.
- Review Section 2.10.4, “Changes in MySQL 8.0” for changes that you should be aware of before upgrading. Some changes may require action.
- Review Section 1.3, “What Is New in MySQL 8.0” for deprecated and removed features. An upgrade may require changes with respect to those features if you use any of them.
- Review Section 1.4, “Server and Status Variables and Options Added, Deprecated, or Removed in MySQL 8.0”. If you use deprecated or removed variables, an upgrade may require configuration changes.
- Review the Release Notes for information about fixes, changes, and new features.
- If you use replication, review Section 17.5.3, “Upgrading a Replication Topology”.
- Upgrade procedures vary by platform and how the initial installation was performed. Use the procedure that applies to your current MySQL installation:
  - For binary and package-based installations on non-Windows platforms, refer to Section 2.10.6, “Upgrading MySQL Binary or Package-based Installations on Unix/Linux”.

    Note

    For supported Linux distributions, the preferred method for upgrading package-based installations is to use the MySQL software repositories (MySQL Yum Repository, MySQL APT Repository, and MySQL SLES Repository).
  - For installations on an Enterprise Linux platform or Fedora using the MySQL Yum Repository, refer to Section 2.10.7, “Upgrading MySQL with the MySQL Yum Repository”.
  - For installations on Ubuntu using the MySQL APT repository, refer to Section 2.10.8, “Upgrading MySQL with the MySQL APT Repository”.
  - For installations on SLES using the MySQL SLES repository, refer to Section 2.10.9, “Upgrading MySQL with the MySQL SLES Repository”.
  - For installations performed using Docker, refer to Section 2.10.11, “Upgrading a Docker Installation of MySQL”.
  - For installations on Windows, refer to Section 2.10.10, “Upgrading MySQL on Windows”.
- If your MySQL installation contains a large amount of data that might take a long time to convert after an in-place upgrade, it may be useful to create a test instance for assessing the conversions that are required and the work involved to perform them. To create a test instance, make a copy of your MySQL instance that contains the `mysql` database and other databases without the data. Run the upgrade procedure on the test instance to assess the work involved to perform the actual data conversion.
- Rebuilding and reinstalling MySQL language interfaces is recommended when you install or upgrade to a new release of MySQL. This applies to MySQL interfaces such as PHP `mysql` extensions and the Perl `DBD::mysql` module.

2.10.2 Upgrade Paths

- Upgrade from MySQL 5.7 to 8.0 is supported. However, upgrade is only supported between General Availability (GA) releases. For MySQL 8.0, it is required that you upgrade from a MySQL 5.7 GA release (5.7.9 or higher). Upgrades from non-GA releases of MySQL 5.7 are not supported.

- Upgrading to the latest release is recommended before upgrading to the next version. For example, upgrade to the latest MySQL 5.7 release before upgrading to MySQL 8.0.
- Upgrade that skips versions is not supported. For example, upgrading directly from MySQL 5.6 to 8.0 is not supported.
- Once a release series reaches General Availability (GA) status, upgrade within the release series (from one GA version to another GA version) is supported. For example, upgrading from MySQL 8.0.*x* to 8.0.*y* is supported. (Upgrade involving development-status non-GA releases is not supported.) Skipping a release is also supported. For example, upgrading from MySQL 8.0.*x* to 8.0.*z* is supported. MySQL 8.0.11 is the first GA status release within the MySQL 8.0 release series.

2.10.3 What the MySQL Upgrade Process Upgrades

Installing a new version of MySQL may require upgrading these parts of the existing installation:

- The `mysql` system schema, which contains tables that store information required by the MySQL server as it runs (see Section 5.3, “The mysql System Schema”). `mysql` schema tables fall into two broad categories:
  - Data dictionary tables, which store database object metadata.
  - System tables (that is, the remaining non-data dictionary tables), which are used for other operational purposes.
- Other schemas, some of which are built in and may be considered “owned” by the server, and others which are not:
  - The Performance Schema, `INFORMATION_SCHEMA`, `ndbinfo`, and `sys` schema.
  - User schemas.

Two distinct version numbers are associated with parts of the installation that may require upgrading:

- The data dictionary version. This applies to the data dictionary tables.
- The server version, also known as the MySQL version. This applies to the system tables and objects in other schemas.

In both cases, the actual version applicable to the existing MySQL installation is stored in the data dictionary, and the current expected version is compiled into the new version of MySQL. When an actual version is lower than the current expected version, those parts of the installation associated with that version must be upgraded to the current version. If both versions indicate an upgrade is needed, the data dictionary upgrade must occur first.

As a reflection of the two distinct versions just mentioned, the upgrade occurs in two steps:

- Step 1: Data dictionary upgrade.

  This step upgrades:

  - The data dictionary tables in the `mysql` schema. If the actual data dictionary version is lower than the current expected version, the server creates data dictionary tables with updated definitions, copies persisted metadata to the new tables, atomically replaces the old tables with the new ones, and reinitializes the data dictionary.
  - The Performance Schema, `INFORMATION_SCHEMA`, and `ndbinfo`.
- Step 2: Server upgrade.

  This step comprises all other upgrade tasks. If the server version of the existing MySQL installation is lower than that of the new installed MySQL version, everything else must be upgraded:

- The system tables in the `mysql` schema (the remaining non-data dictionary tables).
- The `sys` schema.
- User schemas.

The data dictionary upgrade (step 1) is the responsibility of the server, which performs this task as necessary at startup unless invoked with an option that prevents it from doing so. The option is `--upgrade=NONE` as of MySQL 8.0.16, `--no-dd-upgrade` prior to MySQL 8.0.16.

If the data dictionary is out of date but the server is prevented from upgrading it, the server does not run, and exits with an error instead. For example:

```
[ERROR] [MY-013381] [Server] Server shutting down because upgrade is
required, yet prohibited by the command line option '--upgrade=NONE'.
[ERROR] [MY-010334] [Server] Failed to initialize DD Storage Engine
[ERROR] [MY-010020] [Server] Data Dictionary initialization failed.
```

Some changes to the responsibility for step 2 occurred in MySQL 8.0.16:

- Prior to MySQL 8.0.16, `mysql_upgrade` upgrades the Performance Schema, the `INFORMATION_SCHEMA`, and the objects described in step 2. The DBA is expected to invoke `mysql_upgrade` manually after starting the server.
- As of MySQL 8.0.16, the server performs all tasks previously handled by `mysql_upgrade`. Although upgrading remains a two-step operation, the server performs them both, resulting in a simpler process.

Depending on the version of MySQL to which you are upgrading, the instructions in In-Place Upgrade and Logical Upgrade indicate whether the server performs all upgrade tasks or whether you must also invoke `mysql_upgrade` after server startup.

Note

Because the server upgrades the Performance Schema, `INFORMATION_SCHEMA`, and the objects described in step 2 as of MySQL 8.0.16, `mysql_upgrade` is unneeded and is deprecated as of that version; expect it to be removed in a future version of MySQL.

Most aspects of what occurs during step 2 are the same prior to and as of MySQL 8.0.16, although different command options may be needed to achieve a particular effect.

As of MySQL 8.0.16, the `--upgrade` server option controls whether and how the server performs an automatic upgrade at startup:

- With no option or with `--upgrade=AUTO`, the server upgrades anything it determines to be out of date (steps 1 and 2).
- With `--upgrade=NONE`, the server upgrades nothing (skips steps 1 and 2), but also exits with an error if the data dictionary must be upgraded. It is not possible to run the server with an out-of-date data dictionary; the server insists on either upgrading it or exiting.
- With `--upgrade=MINIMAL`, the server upgrades the data dictionary, the Performance Schema, and the `INFORMATION_SCHEMA`, if necessary (step 1). Note that following an upgrade with this option, Group Replication cannot be started, because system tables on which the replication internals depend are not updated, and reduced functionality might also be apparent in other areas.
- With `--upgrade=FORCE`, the server upgrades the data dictionary, the Performance Schema, and the `INFORMATION_SCHEMA`, if necessary (step 1), and forces an upgrade of everything else (step 2). Expect server startup to take longer with this option because the server checks all objects in all schemas.

`FORCE` is useful to force step 2 actions to be performed if the server thinks they are not necessary. One way that `FORCE` differs from `AUTO` is that with `FORCE`, the server re-creates system tables such as help tables or time zone tables if they are missing.

The following list shows upgrade commands prior to MySQL 8.0.16 and the equivalent commands for MySQL 8.0.16 and higher:

- Perform a normal upgrade (steps 1 and 2 as necessary):
  - Prior to MySQL 8.0.16: `mysqld` followed by `mysql_upgrade`
  - As of MySQL 8.0.16: `mysqld`
- Perform only step 1 as necessary:
  - Prior to MySQL 8.0.16: It is not possible to perform all upgrade tasks described in step 1 while excluding those described in step 2. However, you can avoid upgrading user schemas and the `sys` schema using `mysqld` followed by `mysql_upgrade` with the `--upgrade-system-tables` and `--skip-sys-schema` options.
  - As of MySQL 8.0.16: `mysqld --upgrade=MINIMAL`
- Perform step 1 as necessary, and force step 2:
  - Prior to MySQL 8.0.16: `mysqld` followed by `mysql_upgrade --force`
  - As of MySQL 8.0.16: `mysqld --upgrade=FORCE`

Prior to MySQL 8.0.16, certain `mysql_upgrade` options affect the actions it performs. The following table shows which server `--upgrade` option values to use as of MySQL 8.0.16 to achieve similar effects. (These are not necessarily exact equivalents because a given `--upgrade` option value may have additional effects.)

mysql_upgrade Option	Server Option
`--skip-sys-schema`	`--upgrade=NONE` or `--upgrade=MINIMAL`
`--upgrade-system-tables`	`--upgrade=NONE` or `--upgrade=MINIMAL`
`--force`	`--upgrade=FORCE`

Additional notes about what occurs during upgrade step 2:

- Step 2 installs the `sys` schema if it is not installed, and upgrades it to the current version otherwise. An error occurs if a `sys` schema exists but has no `version` view, on the assumption that its absence indicates a user-created schema:

  ```
  A sys schema exists with no sys.version view. If
  you have a user created sys schema, this must be renamed for the
  upgrade to succeed.
  ```

  To upgrade in this case, remove or rename the existing `sys` schema first. Then perform the upgrade procedure again. (It may be necessary to force step 2.)

  To prevent the `sys` schema check:

  - As of MySQL 8.0.16: Start the server with the `--upgrade=NONE` or `--upgrade=MINIMAL` option.
  - Prior to MySQL 8.0.16: Invoke `mysql_upgrade` with the `--skip-sys-schema` option.
- Step 2 upgrades the system tables to ensure that they have the current structure. This is true whether the server or `mysql_upgrade` performs the step. With respect to the content of the help tables and time zone tables, `mysql_upgrade` does not load either type of table, whereas the server loads the help tables, but not the time zone tables. (That is, prior to MySQL 8.0.16, the server loads

the help tables only at data directory initialization time. As of MySQL 8.0.16, it loads the help tables at initialization and upgrade time.) The procedure for loading time zone tables is platform dependent and requires decision making by the DBA, so it cannot be done automatically.

- From MySQL 8.0.30, when Step 2 is upgrading the system tables in the `mysql` schema, the column order in the primary key of the `mysql.db`, `mysql.tables_priv`, `mysql.columns_priv` and `mysql.procs_priv` tables is changed to place the host name and user name columns together. Placing the host name and user name together means that index lookup can be used, which improves performance for `CREATE USER`, `DROP USER`, and `RENAME USER` statements, and for ACL checks for multiple users with multiple privileges. Dropping and re-creating the index is necessary and might take some time if the system has a large number of users and privileges.

- Step 2 processes all tables in all user schemas as necessary. Table checking might take a long time to complete. Each table is locked and therefore unavailable to other sessions while it is being processed. Check and repair operations can be time-consuming, particularly for large tables. Table checking uses the `FOR UPGRADE` option of the `CHECK TABLE` statement. For details about what this option entails, see Section 13.7.3.2, “CHECK TABLE Statement”.

  To prevent table checking:

  - As of MySQL 8.0.16: Start the server with the `--upgrade=NONE` or `--upgrade=MINIMAL` option.
  - Prior to MySQL 8.0.16: Invoke `mysql_upgrade` with the `--upgrade-system-tables` option.

  To force table checking:

  - As of MySQL 8.0.16: Start the server with the `--upgrade=FORCE` option.
  - Prior to MySQL 8.0.16: Invoke `mysql_upgrade` with the `--force` option.

- Step 2 saves the MySQL version number in a file named `mysql_upgrade_info` in the data directory.

  To ignore the `mysql_upgrade_info` file and perform the check regardless:

  - As of MySQL 8.0.16: Start the server with the `--upgrade=FORCE` option.
  - Prior to MySQL 8.0.16: Invoke `mysql_upgrade` with the `--force` option.

  Note

  The `mysql_upgrade_info` file is deprecated; expect it to be removed in a future version of MySQL.

- Step 2 marks all checked and repaired tables with the current MySQL version number. This ensures that the next time upgrade checking occurs with the same version of the server, it can be determined whether there is any need to check or repair a given table again.

2.10.4 Changes in MySQL 8.0

Before upgrading to MySQL 8.0, review the changes described in this section to identify those that apply to your current MySQL installation and applications. Perform any recommended actions.

Changes marked as **Incompatible change** are incompatibilities with earlier versions of MySQL, and may require your attention *before upgrading*. Our aim is to avoid these changes, but occasionally they are necessary to correct problems that would be worse than an incompatibility between releases. If an upgrade issue applicable to your installation involves an incompatibility, follow the instructions given in the description.

- Data Dictionary Changes
- caching_sha2_password as the Preferred Authentication Plugin

Data Dictionary Changes

MySQL Server 8.0 incorporates a global data dictionary containing information about database objects in transactional tables. In previous MySQL series, dictionary data was stored in metadata files and nontransactional system tables. As a result, the upgrade procedure requires that you verify the upgrade readiness of your installation by checking specific prerequisites. For more information, see Section 2.10.5, “Preparing Your Installation for Upgrade”. A data dictionary-enabled server entails some general operational differences; see Section 14.7, “Data Dictionary Usage Differences”.

caching_sha2_password as the Preferred Authentication Plugin

The `caching_sha2_password` and `sha256_password` authentication plugins provide more secure password encryption than the `mysql_native_password` plugin, and `caching_sha2_password` provides better performance than `sha256_password`. Due to these superior security and performance characteristics of `caching_sha2_password`, it is as of MySQL 8.0 the preferred authentication plugin, and is also the default authentication plugin rather than `mysql_native_password`. This change affects both the server and the `libmysqlclient` client library:

- For the server, the default value of the `default_authentication_plugin` system variable changes from `mysql_native_password` to `caching_sha2_password`.

  This change applies only to new accounts created after installing or upgrading to MySQL 8.0 or higher. For accounts already existing in an upgraded installation, their authentication plugin remains unchanged. Existing users who wish to switch to `caching_sha2_password` can do so using the `ALTER USER` statement:

  ```
  ALTER USER user
    IDENTIFIED WITH caching_sha2_password
    BY 'password';
  ```

- The `libmysqlclient` library treats `caching_sha2_password` as the default authentication plugin rather than `mysql_native_password`.

The following sections discuss the implications of the more prominent role of `caching_sha2_password`:

caching_sha2_password Compatibility Issues and Solutions

Important

If your MySQL installation must serve pre-8.0 clients and you encounter compatibility issues after upgrading to MySQL 8.0 or higher, the simplest way to address those issues and restore pre-8.0 compatibility is to reconfigure the server to revert to the previous default authentication plugin (`mysql_native_password`). For example, use these lines in the server option file:

```
[mysqld]
default_authentication_plugin=mysql_native_password
```

That setting enables pre-8.0 clients to connect to 8.0 servers until such time as the clients and connectors in use at your installation are upgraded to know about `caching_sha2_password`. However, the setting should be viewed as temporary, not as a long term or permanent solution, because it causes new accounts created with the setting in effect to forego the improved authentication security provided by `caching_sha2_password`.

The use of `caching_sha2_password` offers more secure password hashing than `mysql_native_password` (and consequent improved client connection authentication). However, it also has compatibility implications that may affect existing MySQL installations:

- Clients and connectors that have not been updated to know about `caching_sha2_password` may have trouble connecting to a MySQL 8.0 server configured with `caching_sha2_password` as the default authentication plugin, even to use accounts that do not authenticate with `caching_sha2_password`. This issue occurs because the server specifies the name of its default authentication plugin to clients. If a client or connector is based on a client/server protocol implementation that does not gracefully handle an unrecognized default authentication plugin, it may fail with an error such as one of these:

  ```
  Authentication plugin 'caching_sha2_password' is not supported
  ```

  ```
  Authentication plugin 'caching_sha2_password' cannot be loaded:
  dlopen(/usr/local/mysql/lib/plugin/caching_sha2_password.so, 2):
  image not found
  ```

  ```
  Warning: mysqli_connect(): The server requested authentication
  method unknown to the client [caching_sha2_password]
  ```

  For information about writing connectors to gracefully handle requests from the server for unknown default authentication plugins, see Authentication Plugin Connector-Writing Considerations.

- Clients that use an account that authenticates with `caching_sha2_password` must use either a secure connection (made using TCP using TLS/SSL credentials, a Unix socket file, or shared memory), or an unencrypted connection that supports password exchange using an RSA key pair. This security requirement does not apply to `mysql_native_passsword`, so the switch to `caching_sha2_password` may require additional configuration (see Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”). However, client connections in MySQL 8.0 prefer use of TLS/SSL by default, so clients that already conform to that preference may need no additional configuration.

- Clients and connectors that have not been updated to know about `caching_sha2_password` *cannot* connect to accounts that authenticate with `caching_sha2_password` because they do not recognize this plugin as valid. (This is a particular instance of how client/server authentication plugin compatibility requirements apply, as discussed at Authentication Plugin Client/Server Compatibility.) To work around this issue, relink clients against `libmysqlclient` from MySQL 8.0 or higher, or obtain an updated connector that recognizes `caching_sha2_password`.

- Because `caching_sha2_password` is also now the default authentication plugin in the `libmysqlclient` client library, authentication requires an extra round trip in the client/server protocol for connections from MySQL 8.0 clients to accounts that use `mysql_native_password` (the previous default authentication plugin), unless the client program is invoked with a `--default-auth=mysql_native_password` option.

The `libmysqlclient` client library for pre-8.0 MySQL versions is able to connect to MySQL 8.0 servers (except for accounts that authenticate with `caching_sha2_password`). That means pre-8.0 clients based on `libmysqlclient` should also be able to connect. Examples:

- Standard MySQL clients such as `mysql` and `mysqladmin` are `libmysqlclient`-based.

- The DBD::mysql driver for Perl DBI is `libmysqlclient`-based.

- MySQL Connector/Python has a C Extension module that is `libmysqlclient`-based. To use it, include the `use_pure=False` option at connect time.

When an existing MySQL 8.0 installation is upgraded to MySQL 8.0.4 or higher, some older `libmysqlclient`-based clients may “automatically” upgrade if they are dynamically linked, because they use the new client library installed by the upgrade. For example, if the DBD::mysql driver for Perl DBI uses dynamic linking, it can use the `libmysqlclient` in place after an upgrade to MySQL 8.0.4 or higher, with this result:

- Prior to the upgrade, DBI scripts that use DBD::mysql can connect to a MySQL 8.0 server, except for accounts that authenticate with `caching_sha2_password`.
- After the upgrade, the same scripts become able to use `caching_sha2_password` accounts as well.

However, the preceding results occur because `libmysqlclient` instances from MySQL 8.0 installations prior to 8.0.4 are binary compatible: They both use a shared library major version number of 21. For clients linked to `libmysqlclient` from MySQL 5.7 or older, they link to a shared library with a different version number that is not binary compatible. In this case, the client must be recompiled against `libmysqlclient` from 8.0.4 or higher for full compatibility with MySQL 8.0 servers and `caching_sha2_password` accounts.

MySQL Connector/J 5.1 through 8.0.8 is able to connect to MySQL 8.0 servers, except for accounts that authenticate with `caching_sha2_password`. (Connector/J 8.0.9 or higher is required to connect to `caching_sha2_password` accounts.)

Clients that use an implementation of the client/server protocol other than `libmysqlclient` may need to be upgraded to a newer version that understands the new authentication plugin. For example, in PHP, MySQL connectivity usually is based on `mysqlnd`, which currently does not know about `caching_sha2_password`. Until an updated version of `mysqlnd` is available, the way to enable PHP clients to connect to MySQL 8.0 is to reconfigure the server to revert to `mysql_native_password` as the default authentication plugin, as previously discussed.

If a client or connector supports an option to explicitly specify a default authentication plugin, use it to name a plugin other than `caching_sha2_password`. Examples:

- Some MySQL clients support a `--default-auth` option. (Standard MySQL clients such as `mysql` and `mysqladmin` support this option but can successfully connect to 8.0 servers without it. However, other clients may support a similar option. If so, it is worth trying it.)
- Programs that use the `libmysqlclient` C API can call the `mysql_options()` function with the `MYSQL_DEFAULT_AUTH` option.
- MySQL Connector/Python scripts that use the native Python implementation of the client/server protocol can specify the `auth_plugin` connection option. (Alternatively, use the Connector/Python C Extension, which is able to connect to MySQL 8.0 servers without the need for `auth_plugin`.)

caching_sha2_password-Compatible Clients and Connectors

If a client or connector is available that has been updated to know about `caching_sha2_password`, using it is the best way to ensure compatibility when connecting to a MySQL 8.0 server configured with `caching_sha2_password` as the default authentication plugin.

These clients and connectors have been upgraded to support `caching_sha2_password`:

- The `libmysqlclient` client library in MySQL 8.0 (8.0.4 or higher). Standard MySQL clients such as `mysql` and `mysqladmin` are `libmysqlclient`-based, so they are compatible as well.
- The `libmysqlclient` client library in MySQL 5.7 (5.7.23 or higher). Standard MySQL clients such as `mysql` and `mysqladmin` are `libmysqlclient`-based, so they are compatible as well.
- MySQL Connector/C++ 1.1.11 or higher or 8.0.7 or higher.

- MySQL Connector/J 8.0.9 or higher.
- MySQL Connector/NET 8.0.10 or higher (through the classic MySQL protocol).
- MySQL Connector/Node.js 8.0.9 or higher.
- PHP: the X DevAPI PHP extension (mysql_xdevapi) supports `caching_sha2_password`.

  PHP: the PDO_MySQL and ext/mysqli extensions do not support `caching_sha2_password`. In addition, when used with PHP versions before 7.1.16 and PHP 7.2 before 7.2.4, they fail to connect with `default_authentication_plugin=caching_sha2_password` even if `caching_sha2_password` is not used.

caching_sha2_password and the root Administrative Account

For upgrades to MySQL 8.0, the authentication plugin existing accounts remains unchanged, including the plugin for the `'root'@'localhost'` administrative account.

For new MySQL 8.0 installations, when you initialize the data directory (using the instructions at Section 2.9.1, “Initializing the Data Directory”), the `'root'@'localhost'` account is created, and that account uses `caching_sha2_password` by default. To connect to the server following data directory initialization, you must therefore use a client or connector that supports `caching_sha2_password`. If you can do this but prefer that the `root` account use `mysql_native_password` after installation, install MySQL and initialize the data directory as you normally would. Then connect to the server as `root` and use `ALTER USER` as follows to change the account authentication plugin and password:

```
ALTER USER 'root'@'localhost'
  IDENTIFIED WITH mysql_native_password
  BY 'password';
```

If the client or connector that you use does not yet support `caching_sha2_password`, you can use a modified data directory-initialization procedure that associates the `root` account with `mysql_native_password` as soon as the account is created. To do so, use either of these techniques:

- Supply a `--default-authentication-plugin=mysql_native_password` option along with `--initialize` or `--initialize-insecure`.
- Set `default_authentication_plugin` to `mysql_native_password` in an option file, and name that option file using a `--defaults-file` option along with `--initialize` or `--initialize-insecure`. (In this case, if you continue to use that option file for subsequent server startups, new accounts are created with `mysql_native_password` rather than `caching_sha2_password` unless you remove the `default_authentication_plugin` setting from the option file.)

caching_sha2_password and Replication

In replication scenarios for which all servers have been upgraded to MySQL 8.0.4 or higher, replica connections to source servers can use accounts that authenticate with `caching_sha2_password`. For such connections, the same requirement applies as for other clients that use accounts that authenticate with `caching_sha2_password`: Use a secure connection or RSA-based password exchange.

To connect to a `caching_sha2_password` account for source/replica replication:

- Use any of the following `CHANGE MASTER TO` options:

```
MASTER_SSL = 1
GET_MASTER_PUBLIC_KEY = 1
MASTER_PUBLIC_KEY_PATH='path to RSA public key file'
```

- Alternatively, you can use the RSA public key-related options if the required keys are supplied at server startup.

To connect to a `caching_sha2_password` account for Group Replication:

- For MySQL built using OpenSSL, set any of the following system variables:

```
SET GLOBAL group_replication_recovery_use_ssl = ON;
SET GLOBAL group_replication_recovery_get_public_key = 1;
SET GLOBAL group_replication_recovery_public_key_path = 'path to RSA public key file';
```

- Alternatively, you can use the RSA public key-related options if the required keys are supplied at server startup.

Configuration Changes

- **Incompatible change**: A MySQL storage engine is now responsible for providing its own partitioning handler, and the MySQL server no longer provides generic partitioning support. `InnoDB` and `NDB` are the only storage engines that provide a native partitioning handler that is supported in MySQL 8.0. A partitioned table using any other storage engine must be altered—either to convert it to `InnoDB` or `NDB`, or to remove its partitioning—*before* upgrading the server, else it cannot be used afterwards.

  For information about converting `MyISAM` tables to `InnoDB`, see Section 15.6.1.5, “Converting Tables from MyISAM to InnoDB”.

  A table creation statement that would result in a partitioned table using a storage engine without such support fails with an error (`ER_CHECK_NOT_IMPLEMENTED`) in MySQL 8.0. If you import databases from a dump file created in MySQL 5.7 (or earlier) using `mysqldump` into a MySQL 8.0 server, you must make sure that any statements creating partitioned tables do not also specify an unsupported storage engine, either by removing any references to partitioning, or by specifying the storage engine as `InnoDB` or allowing it to be set as `InnoDB` by default.

  Note

  The procedure given at Section 2.10.5, “Preparing Your Installation for Upgrade”, describes how to identify partitioned tables that must be altered before upgrading to MySQL 8.0.

  See Section 24.6.2, “Partitioning Limitations Relating to Storage Engines”, for further information.

- **Incompatible change**: Several server error codes are not used and have been removed (for a list, see Features Removed in MySQL 8.0). Applications that test specifically for any of them should be updated.

- **Important change**: The default character set has changed from `latin1` to `utf8mb4`. These system variables are affected:

  - The default value of the `character_set_server` and `character_set_database` system variables has changed from `latin1` to `utf8mb4`.

  - The default value of the `collation_server` and `collation_database` system variables has changed from `latin1_swedish_ci` to `utf8mb4_0900_ai_ci`.

  As a result, the default character set and collation for new objects differ from previously unless an explicit character set and collation are specified. This includes databases and objects within them, such as tables, views, and stored programs. Assuming that the previous defaults were used, one way to preserve them is to start the server with these lines in the `my.cnf` file:

```
[mysqld]
character_set_server=latin1
collation_server=latin1_swedish_ci
```

  In a replicated setting, when upgrading from MySQL 5.7 to 8.0, it is advisable to change the default character set back to the character set used in MySQL 5.7 before upgrading. After the upgrade is completed, the default character set can be changed to `utf8mb4`.

- **Incompatible change**: As of MySQL 8.0.11, it is prohibited to start the server with a `lower_case_table_names` setting that is different from the setting used when the server was initialized. The restriction is necessary because collations used by various data dictionary table fields are based on the `lower_case_table_names` setting that was defined when the server was initialized, and restarting the server with a different setting would introduce inconsistencies with respect to how identifiers are ordered and compared.

Server Changes

- In MySQL 8.0.11, several deprecated features related to account management have been removed, such as use of the `GRANT` statement to modify nonprivilege characteristics of user accounts, the `NO_AUTO_CREATE_USER` SQL mode, the `PASSWORD()` function, and the `old_passwords` system variable.

  Replication from MySQL 5.7 to 8.0 of statements that refer to these removed features can cause replication failure. Applications that use any of the removed features should be revised to avoid them and use alternatives when possible, as described in Features Removed in MySQL 8.0.

  To avoid a startup failure on MySQL 8.0, remove any instance of `NO_AUTO_CREATE_USER` from `sql_mode` system variable settings in MySQL option files.

  Loading a dump file that includes the `NO_AUTO_CREATE_USER` SQL mode in stored program definitions into a MySQL 8.0 server causes a failure. As of MySQL 5.7.24 and MySQL 8.0.13, `mysqldump` removes `NO_AUTO_CREATE_USER` from stored program definitions. Dump files created with an earlier version of `mysqldump` must be modified manually to remove instances of `NO_AUTO_CREATE_USER`.

- In MySQL 8.0.11, these deprecated compatibility SQL modes were removed: `DB2`, `MAXDB`, `MSSQL`, `MYSQL323`, `MYSQL40`, `ORACLE`, `POSTGRESQL`, `NO_FIELD_OPTIONS`, `NO_KEY_OPTIONS`, `NO_TABLE_OPTIONS`. They can no longer be assigned to the `sql_mode` system variable or used as permitted values for the `mysqldump --compatible` option.

  Removal of `MAXDB` means that the `TIMESTAMP` data type for `CREATE TABLE` or `ALTER TABLE` is no longer treated as `DATETIME`.

  Replication from MySQL 5.7 to 8.0 of statements that refer to the removed SQL modes can cause replication failure. This includes replication of `CREATE` statements for stored programs (stored procedures and functions, triggers, and events) that are executed while the current `sql_mode` value includes any of the removed modes. Applications that use any of the removed modes should be revised to avoid them.

- As of MySQL 8.0.3, spatial data types permit an `SRID` attribute, to explicitly indicate the spatial reference system (SRS) for values stored in the column. See Section 11.4.1, “Spatial Data Types”.

  A spatial column with an explicit `SRID` attribute is SRID-restricted: The column takes only values with that ID, and `SPATIAL` indexes on the column become subject to use by the optimizer. The optimizer ignores `SPATIAL` indexes on spatial columns with no `SRID` attribute. See Section 8.3.3, “SPATIAL Index Optimization”. If you want the optimizer to consider `SPATIAL` indexes on spatial columns that are not SRID-restricted, each such column should be modified:

  - Verify that all values within the column have the same SRID. To determine the SRIDs contained in a geometry column *`col_name`*, use the following query:

    ```
    SELECT DISTINCT ST_SRID(col_name) FROM tbl_name;
    ```

    If the query returns more than one row, the column contains a mix of SRIDs. In that case, modify its contents so all values have the same SRID.

  - Redefine the column to have an explicit `SRID` attribute.

  - Recreate the `SPATIAL` index.

- Several spatial functions were removed in MySQL 8.0.0 due to a spatial function namespace change that implemented an `ST_` prefix for functions that perform an exact operation, or an `MBR` prefix for functions that perform an operation based on minimum bounding rectangles. The use of removed spatial functions in generated column definitions could cause an upgrade failure. Before upgrading, run `mysqlcheck --check-upgrade` for removed spatial functions and replace any that you find with their `ST_` or `MBR` named replacements. For a list of removed spatial functions, refer to Features Removed in MySQL 8.0.

- The `BACKUP_ADMIN` privilege is automatically granted to users with the `RELOAD` privilege when performing an in-place upgrade to MySQL 8.0.3 or higher.

- From MySQL 8.0.13, because of differences between row-based or mixed replication mode and statement-based replication mode in the way that temporary tables are handled, there are new restrictions on switching the binary logging format at runtime.

  - `SET @@SESSION.binlog_format` cannot be used if the session has any open temporary tables.

  - `SET @@global.binlog_format` and `SET @@persist.binlog_format` cannot be used if any replication channel has any open temporary tables. `SET @@persist_only.binlog_format` is allowed if replication channels have open temporary tables, because unlike `PERSIST`, `PERSIST_ONLY` does not modify the runtime global system variable value.

  - `SET @@global.binlog_format` and `SET @@persist.binlog_format` cannot be used if any replication channel applier is running. This is because the change only takes effect on a replication channel when its applier is restarted, at which time the replication channel might have open temporary tables. This behavior is more restrictive than before. `SET @@persist_only.binlog_format` is allowed if any replication channel applier is running.

  - From MySQL 8.0.27, configuring a session setting for `internal_tmp_mem_storage_engine` requires the `SESSION_VARIABLES_ADMIN` or `SYSTEM_VARIABLES_ADMIN` privilege.

  - As of MySQL 8.0.27, the clone plugin permits concurrent DDL operations on the donor MySQL Server instance while a cloning operation is in progress. Previously, a backup lock was held during the cloning operation, preventing concurrent DDL on the donor. To revert to the previous behavior of blocking concurrent DDL on the donor during a clone operation, enable the `clone_block_ddl` variable. See Section 5.6.7.4, “Cloning and Concurrent DDL”.

- From MySQL 8.0.30, error log components listed in the `log_error_services` value at startup are loaded implicitly early in the MySQL Server startup sequence. If you have previously installed loadable error log components using `INSTALL COMPONENT` and you list those components in a `log_error_services` setting that is read at startup (from an option file, for example), your configuration should be updated to avoid startup warnings. For more information, see Error Log Configuration Methods.

InnoDB Changes

- `INFORMATION_SCHEMA` views based on `InnoDB` system tables were replaced by internal system views on data dictionary tables. Affected `InnoDB` `INFORMATION_SCHEMA` views were renamed:

  Table 2.16 Renamed InnoDB Information Schema Views

Old Name	New Name
`INNODB_SYS_COLUMNS`	`INNODB_COLUMNS`
`INNODB_SYS_DATAFILES`	`INNODB_DATAFILES`
`INNODB_SYS_FIELDS`	`INNODB_FIELDS`
`INNODB_SYS_FOREIGN`	`INNODB_FOREIGN`
`INNODB_SYS_FOREIGN_COLS`	`INNODB_FOREIGN_COLS`

Old Name	New Name
INNODB_SYS_INDEXES	INNODB_INDEXES
INNODB_SYS_TABLES	INNODB_TABLES
INNODB_SYS_TABLESPACES	INNODB_TABLESPACES
INNODB_SYS_TABLESTATS	INNODB_TABLESTATS
INNODB_SYS_VIRTUAL	INNODB_VIRTUAL

After upgrading to MySQL 8.0.3 or higher, update any scripts that reference previous InnoDB INFORMATION_SCHEMA view names.

- The zlib library version bundled with MySQL was raised from version 1.2.3 to version 1.2.11.

  The zlib compressBound() function in zlib 1.2.11 returns a slightly higher estimate of the buffer size required to compress a given length of bytes than it did in zlib version 1.2.3. The compressBound() function is called by InnoDB functions that determine the maximum row size permitted when creating compressed InnoDB tables or inserting and updating rows in compressed InnoDB tables. As a result, CREATE TABLE ... ROW_FORMAT=COMPRESSED, INSERT, and UPDATE operations with row sizes very close to the maximum row size that were successful in earlier releases could now fail. To avoid this issue, test CREATE TABLE statements for compressed InnoDB tables with large rows on a MySQL 8.0 test instance prior to upgrading.

- With the introduction of the --innodb-directories feature, the location of file-per-table and general tablespace files created with an absolute path or in a location outside of the data directory should be added to the innodb_directories argument value. Otherwise, InnoDB is not able to locate these files during recovery. To view tablespace file locations, query the Information Schema FILES table:

  ```
  SELECT TABLESPACE_NAME, FILE_NAME FROM INFORMATION_SCHEMA.FILES \G
  ```

- Undo logs can no longer reside in the system tablespace. In MySQL 8.0, undo logs reside in two undo tablespaces by default. For more information, see Section 15.6.3.4, “Undo Tablespaces”.

  When upgrading from MySQL 5.7 to MySQL 8.0, any undo tablespaces that exist in the MySQL 5.7 instance are removed and replaced by two new default undo tablespaces. Default undo tablespaces are created in the location defined by the innodb_undo_directory variable. If the innodb_undo_directory variable is undefined, undo tablespaces are created in the data directory. Upgrade from MySQL 5.7 to MySQL 8.0 requires a slow shutdown which ensures that undo tablespaces in the MySQL 5.7 instance are empty, permitting them to be removed safely.

  When upgrading to MySQL 8.0.14 or later from an earlier MySQL 8.0 release, undo tablespaces that exist in the pre-upgrade instance as a result of an innodb_undo_tablespaces setting greater than 2 are treated as user-defined undo tablespaces, which can be deactivated and dropped using ALTER UNDO TABLESPACE and DROP UNDO TABLESPACE syntax, respectively, after upgrading. Upgrade within the MySQL 8.0 release series may not always require a slow shutdown which means that existing undo tablespaces could contain undo logs. Therefore, existing undo tablespaces are not removed by the upgrade process.

- **Incompatible change**: As of MySQL 8.0.17, the CREATE TABLESPACE ... ADD DATAFILE clause does not permit circular directory references. For example, the circular directory reference (/../) in the following statement is not permitted:

  ```
  CREATE TABLESPACE ts1 ADD DATAFILE ts1.ibd 'any_directory/../ts1.ibd';
  ```

  An exception to the restriction exists on Linux, where a circular directory reference is permitted if the preceding directory is a symbolic link. For example, the data file path in the example above is

permitted if *any_directory* is a symbolic link. (It is still permitted for data file paths to begin with '../'.)

To avoid upgrade issues, remove any circular directory references from tablespace data file paths before upgrading to MySQL 8.0.17 or higher. To inspect tablespace paths, query the Information Schema INNODB_DATAFILES table.

- Due to a regression introduced in MySQL 8.0.14, in-place upgrade on a case-sensitive file system from MySQL 5.7 or a MySQL 8.0 release prior to MySQL 8.0.14 to MySQL 8.0.16 failed for instances with partitioned tables and lower_case_table_names=1. The failure was caused by a case mismatch issue related to partitioned table file names. The fix that introduced the regression was reverted, which permits upgrades to MySQL 8.0.17 from MySQL 5.7 or MySQL 8.0 releases prior to MySQL 8.0.14 to function as normal. However, the regression is still present in the MySQL 8.0.14, 8.0.15, and 8.0.16 releases.

  In-place upgrade on a case-sensitive file system from MySQL 8.0.14, 8.0.15, or 8.0.16 to MySQL 8.0.17 fails with the following error when starting the server after upgrading binaries or packages to MySQL 8.0.17 if partitioned tables are present and lower_case_table_names=1:

  ```
  Upgrading from server version version_number with
  partitioned tables and lower_case_table_names == 1 on a case sensitive file
  system may cause issues, and is therefore prohibited. To upgrade anyway, restart
  the new server version with the command line option 'upgrade=FORCE'. When
  upgrade is completed, please execute 'RENAME TABLE part_table_name
  TO new_table_name; RENAME TABLE new_table_name
  TO part_table_name;' for each of the partitioned tables.
  Please see the documentation for further information.
  ```

  If you encounter this error when upgrading to MySQL 8.0.17, perform the following workaround:

  1. Restart the server with --upgrade=force to force the upgrade operation to proceed.
  2. Identify partitioned table file names with lowercase partition name delimiters (#p# or #sp#):

     ```
     mysql> SELECT FILE_NAME FROM INFORMATION_SCHEMA.FILES WHERE FILE_NAME LIKE '%#p#%' OR FILE_NAME LIKE
     ```

  3. For each file identified, rename the associated table using a temporary name, then rename the table back to its original name.

     ```
     mysql> RENAME TABLE table_name TO temporary_table_name;
     mysql> RENAME TABLE temporary_table_name TO table_name;
     ```

  4. Verify that there are no partitioned table file names lowercase partition name delimiters (an empty result set should be returned).

     ```
     mysql> SELECT FILE_NAME FROM INFORMATION_SCHEMA.FILES WHERE FILE_NAME LIKE '%#p#%' OR FILE_NAME LIKE
     Empty set (0.00 sec)
     ```

  5. Run ANALYZE TABLE on each renamed table to update the optimizer statistics in the mysql.innodb_index_stats and mysql.innodb_table_stats tables.

  Because of the regression still present in the MySQL 8.0.14, 8.0.15, and 8.0.16 releases, importing partitioned tables from MySQL 8.0.14, 8.0.15, or 8.0.16 to MySQL 8.0.17 is not supported on case-sensitive file systems where lower_case_table_names=1. Attempting to do so results in a "Tablespace is missing for table" error.

- MySQL uses delimiter strings when constructing tablespace names and file names for table partitions. A " #p# " delimiter string precedes partition names, and an " #sp# " delimiter string precedes subpartition names, as shown:

  ```
  schema_name.table_name#p#partition_name#sp#subpartition_name
  ```

```
table_name#p#partition_name#sp#subpartition_name.ibd
```

Historically, delimiter strings have been uppercase (#P# and #SP#) on case-sensitive file systems such as Linux, and lowercase (#p# and #sp#) on case-insensitive file systems such as Windows. As of MySQL 8.0.19, delimiter strings are lowercase on all file systems. This change prevents issues when migrating data directories between case-sensitive and case-insensitive file systems. Uppercase delimiter strings are no longer used.

Additionally, partition tablespace names and file names generated based on user-specified partition or subpartition names, which can be specified in uppercase or lowercase, are now generated (and stored internally) in lowercase regardless of the lower_case_table_names setting to ensure case-insensitivity. For example, if a table partition is created with the name PART_1, the tablespace name and file name are generated in lowercase:

```
schema_name.table_name#p#part_1
table_name#p#part_1.ibd
```

During upgrade, MySQL checks and modifies if necessary:

- Partition file names on disk and in the data dictionary to ensure lowercase delimiters and partition names.
- Partition metadata in the data dictionary for related issues introduced by previous bug fixes.
- InnoDB statistics data for related issues introduced by previous bug fixes.

During tablespace import operations, partition tablespace file names on disk are checked and modified if necessary to ensure lowercase delimiters and partition names.

- As of MySQL 8.0.21, a warning is written to the error log at startup or when upgrading from MySQL 5.7 if tablespace data files are found to reside in unknown directories. Known directories are those defined by the datadir, innodb_data_home_dir, and innodb_directories variables. To make a directory known, add it to the innodb_directories setting. Making directories known ensures that data files can be found during recovery. For more information, see Tablespace Discovery During Crash Recovery.

- **Important change**: From MySQL 8.0.30, the innodb_redo_log_capacity variable controls the amount of disk space occupied by redo log files. With this change, the default number of redo log files and their location has also changed. From MySQL 8.0.30, InnoDB maintains 32 redo log files in the #innodb_redo directory in the data directory. Previously, InnoDB created two redo log files in the data directory by default, and the number and size of redo log files were controlled by the innodb_log_files_in_group and innodb_log_file_size variables. These two variables are now deprecated.

  When the innodb_redo_log_capacity setting is defined, innodb_log_files_in_group and innodb_log_file_size settings are ignored; otherwise, those settings are used to compute the innodb_redo_log_capacity setting (innodb_log_files_in_group * innodb_log_file_size = innodb_redo_log_capacity). If none of those variables are set, redo log capacity is set to the innodb_redo_log_capacity default value, which is 104857600 bytes (100MB).

  As is generally required for any upgrade, this change requires a clean shutdown before upgrading.

  For more information about this feature, see Section 15.6.5, “Redo Log”.

- Before MySQL 5.7.35, there was no size limitation for indexes in tables with redundant or compact row format. As of MySQL 5.7.35, the limit is 767 bytes. An upgrade from a MySQL version before 5.7.35 to MySQL 8.0 can produce inaccessible tables. If a table with redundant or compact row

format has an index larger than 767 bytes, drop the index and re-create it before an upgrade to MySQL 8.0. The error message is:

```
mysql> ERROR 1709 (HY000): Index column size too large. The maximum column size is 767 bytes.
```

SQL Changes

- **Incompatible change**: As of MySQL 8.0.13, the deprecated `ASC` or `DESC` qualifiers for `GROUP BY` clauses have been removed. Queries that previously relied on `GROUP BY` sorting may produce results that differ from previous MySQL versions. To produce a given sort order, provide an `ORDER BY` clause.

  Queries and stored program definitions from MySQL 8.0.12 or lower that use `ASC` or `DESC` qualifiers for `GROUP BY` clauses should be amended. Otherwise, upgrading to MySQL 8.0.13 or higher may fail, as may replicating to MySQL 8.0.13 or higher replica servers.

- Some keywords may be reserved in MySQL 8.0 that were not reserved in MySQL 5.7. See Section 9.3, “Keywords and Reserved Words”. This can cause words previously used as identifiers to become illegal. To fix affected statements, use identifier quoting. See Section 9.2, “Schema Object Names”.

- After upgrading, it is recommended that you test optimizer hints specified in application code to ensure that the hints are still required to achieve the desired optimization strategy. Optimizer enhancements can sometimes render certain optimizer hints unnecessary. In some cases, an unnecessary optimizer hint may even be counterproductive.

- **Incompatible change**: In MySQL 5.7, specifying a `FOREIGN KEY` definition for an `InnoDB` table without a `CONSTRAINT` *`symbol`* clause, or specifying the `CONSTRAINT` keyword without a `symbol`, causes `InnoDB` to use a generated constraint name. That behavior changed in MySQL 8.0, with `InnoDB` using the `FOREIGN KEY` *`index_name`* value instead of a generated name. Because constraint names must be unique per schema (database), the change caused errors due to foreign key index names that were not unique per schema. To avoid such errors, the new constraint naming behavior has been reverted in MySQL 8.0.16, and `InnoDB` once again uses a generated constraint name.

  For consistency with `InnoDB`, `NDB` releases based on MySQL 8.0.16 or higher use a generated constraint name if the `CONSTRAINT` *`symbol`* clause is not specified, or the `CONSTRAINT` keyword is specified without a `symbol`. `NDB` releases based on MySQL 5.7 and earlier MySQL 8.0 releases used the `FOREIGN KEY` *`index_name`* value.

  The changes described above may introduce incompatibilities for applications that depend on the previous foreign key constraint naming behavior.

- The handling of system variable values by MySQL flow control functions such as `IFNULL()` and `CASE()` changed in MySQL 8.0.22; system variable values are now handled as column values of the same character and collation, rather than as constants. Some queries using these functions with system variables that were previously successful may subsequently be rejected with `Illegal mix of collations`. In such cases, cast the system variable to the correct character set and collation.

- **Incompatible change**: MySQL 8.0.28 fixes an issue in previous MySQL 8.0 releases whereby the `CONVERT()` function sometimes allowed invalid casts of `BINARY` values to nonbinary character sets. Applications which may have relied on this behavior should be checked and if necessary modified prior to upgrade.

  In particular, where `CONVERT()` was used as part of an expression for an indexed generated column, the change in the function's behavior may result in index corruption following an upgrade to MySQL 8.0.28. You can prevent this from happening by following these steps:

  1. Prior to performing the upgrade, correct any invalid input data.

  2. Drop and then re-create the index.

You can also force a table rebuild using `ALTER TABLE table FORCE`, instead.

3. Upgrade the MySQL software.

If you cannot validate the input data beforehand, you should not re-create the index or rebuild the table until after you perform the upgrade to MySQL 8.0.28.

Changed Server Defaults

MySQL 8.0 comes with improved defaults, aiming at the best out of the box experience possible. These changes are driven by the fact that technology is advancing (machines have more CPUS, use SSDs and so on), more data is being stored, MySQL is evolving (InnoDB, Group Replication, AdminAPI), and so on. The following table summarizes the defaults which have been changed to provide the best MySQL experience for the majority of users.

Option/Parameter	Old Default	New Default
Server changes		
`character_set_server`	latin1	utf8mb4
`collation_server`	latin1_swedish_ci	utf8mb4_0900_ai_ci
`explicit_defaults_for_timestamp`	OFF	ON
`optimizer_trace_max_mem_size`	16KB	1MB
`validate_password_check_user_name`	OFF	ON
`back_log`	-1 (autosize) changed from : back_log = 50 + (max_connections / 5)	-1 (autosize) changed to : back_log = max_connections
`max_allowed_packet`	4194304 (4MB)	67108864 (64MB)
`max_error_count`	64	1024
`event_scheduler`	OFF	ON
`table_open_cache`	2000	4000
`log_error_verbosity`	3 (Notes)	2 (Warning)
`local_infile`	ON (5.7)	OFF
InnoDB changes		
`innodb_undo_tablespaces`	0	2
`innodb_undo_log_truncate`	OFF	ON
`innodb_flush_method`	NULL	fsync (Unix), unbuffered (Windows)
`innodb_autoinc_lock_mode`	1 (consecutive)	2 (interleaved)
`innodb_flush_neighbors`	1 (enable)	0 (disable)
`innodb_max_dirty_pages_pct_lwm`	0 (%)	10 (%)
`innodb_max_dirty_pages_pct`	75 (%)	90 (%)
Performance Schema changes		
`performance-schema-instrument='wait/lock/metadata/sql/%=ON'`	OFF	ON
`performance-schema-instrument='memory/%=COUNTED'`	OFF	COUNTED

Option/Parameter	Old Default	New Default
performance-schema-consumer-events-transactions-current=ON	OFF	ON
performance-schema-consumer-events-transactions-history=ON	OFF	ON
performance-schema-instrument='transaction%=ON'	OFF	ON
Replication changes		
log_bin	OFF	ON
server_id	0	1
log-slave-updates	OFF	ON
expire_logs_days	0	30
master-info-repository	FILE	TABLE
relay-log-info-repository	FILE	TABLE
transaction-write-set-extraction	OFF	XXHASH64
slave_rows_search_algorithms	INDEX_SCAN, TABLE_SCAN	INDEX_SCAN, HASH_SCAN
slave_pending_jobs_size_max	16M	128M
gtid_executed_compression_period	1000	0
Group Replication changes		
group_replication_autorejoin_tries	0	3
group_replication_exit_state_action	ABORT_SERVER	READ_ONLY
group_replication_member_expel_timeout	0	5

For more information about options or variables which have been added, see Option/Variable Changes for MySQL 8.0, in the *MySQL Server Version Reference*.

The following sections explain the changes to defaults and any impact they might have on your deployment.

Server Defaults

- The default value of the character_set_server system variable and command line option --character-set-server changed from latin1 to utf8mb4. This is the server's default character set. At this time, UTF8MB4 is the dominant character encoding for the web, and this change makes life easier for the vast majority of MySQL users. The upgrade from 5.7 to 8.0 does not change the character set for any existing database objects, but, unless you set character_set_server explicitly (either back to the previous value, or to a new one), a new schema uses utf8mb4 by default. We recommend that you move to utf8mb4 whenever possible.
- The default value of the collation_server system variable and command line argument --collation-server changed from latin1_swedish_ci to utf8mb4_0900_ai_ci. This is the server's default collation, the ordering of characters in a character set. There is a link between collations and character sets as each character set comes with a list of possible collations. The upgrade from 5.7 to 8.0 does not change any collation for any existing database objects, but takes effect for new objects.

- The default value of the `explicit_defaults_for_timestamp` system variable changed from `OFF` (MySQL legacy behavior) to `ON` (SQL standard behavior). This option was originally introduced in 5.6 and was `OFF` in 5.6 and 5.7.
- The default value of the `optimizer_trace_max_mem_size` system variable changed from `16KB` to `1MB`. The old default caused the optimizer trace to be truncated for any non-trivial query. This change ensures useful optimizer traces for most queries.
- The default value of the `validate_password_check_user_name` system variable changed from `OFF` to `ON`. This means that when the `validate_password` plugin is enabled, by default it now rejects passwords that match the current session user name.
- The autosize algorithm for the `back_log` system variable has changed. The value for autosize (-1) is now set to the value of `max_connections`, which is bigger than the calculated by `50 + (max_connections / 5)`. The `back_log` queues up incoming IP connect requests in situations where the server is not able to keep up with incoming requests. In the worst case, with `max_connections` number of clients trying to reconnect at the same time, for example after a network failure, they can all be buffered and reject-retry loops are avoided.
- The default value of the `max_allowed_packet` system variable changed from `4194304` (4M) to `67108864` (64M). The main advantage with this larger default is less chance of receiving errors about an insert or query being larger than `max_allowed_packet`. It should be as big as the largest Section 11.3.4, “The BLOB and TEXT Types” you want to use. To revert to the previous behavior, set `max_allowed_packet=4194304`.
- The default value of the `max_error_count` system variable changed from `64` to `1024`. This ensures that MySQL handles a larger number of warnings, such as an UPDATE statement that touches 1000s of rows and many of them give conversion warnings. It is common for many tools to batch updates, to help reduce replication lag. External tools such as pt-online-schema-change defaults to 1000, and gh-ost defaults to 100. MySQL 8.0 covers full error history for these two use cases. There are no static allocations, so this change only affects memory consumption for statements that generate lots of warnings.
- The default value of the `event_scheduler` system variable changed from `OFF` to `ON`. In other words, the event scheduler is enabled by default. This is an enabler for new features in SYS, for example “kill idle transactions”.
- The default value of the `table_open_cache` system variable changed from `2000` to `4000`. This is a minor change which increases session concurrency on table access.
- The default value of the `log_error_verbosity` system variable changed from `3` (Notes) to `2` (Warning). The purpose is to make the MySQL 8.0 error log less verbose by default.

InnoDB Defaults

- **Incompatible change** The default value of the `innodb_undo_tablespaces` system variable changed from `0` to `2`. The configures the number of undo tablespaces used by InnoDB. In MySQL 8.0 the minimum value for `innodb_undo_tablespaces` is 2 and rollback segments cannot be created in the system tablespace anymore. Thus, this is a case where you cannot revert back to 5.7 behavior. The purpose of this change is to be able to auto-truncate Undo logs (see next item), reclaiming disk space used by (occasional) long transactions such as a `mysqldump`.
- The default value of the `innodb_undo_log_truncate` system variable changed from `OFF` to `ON`. When enabled, undo tablespaces that exceed the threshold value defined by `innodb_max_undo_log_size` are marked for truncation. Only undo tablespaces can be truncated. Truncating undo logs that reside in the system tablespace is not supported. An upgrade from 5.7 to 8.0 automatically converts your system to use undo tablespaces, using the system tablespace is not an option in 8.0.
- The default value of the `innodb_flush_method` system variable changed from `NULL` to `fsync` on Unix-like systems and from `NULL` to `unbuffered` on Windows systems. This is

more of a terminology and option cleanup without any tangible impact. For Unix this is just a documentation change as the default was `fsync` also in 5.7 (the default `NULL` meant `fsync`). Similarly on Windows, `innodb_flush_method` default `NULL` meant `async_unbuffered` in 5.7, and is replaced by default `unbuffered` in 8.0, which in combination with the existing default `innodb_use_native_aio=ON` has the same effect.

- **Incompatible change** The default value of the `innodb_autoinc_lock_mode` system variable changed from `1` (consecutive) to `2` (interleaved). The change to interleaved lock mode as the default setting reflects the change from statement-based to row-based replication as the default replication type, which occurred in MySQL 5.7. *Statement-based replication* requires the consecutive auto-increment lock mode to ensure that auto-increment values are assigned in a predictable and repeatable order for a given sequence of SQL statements, whereas *row-based replication* is not sensitive to the execution order of SQL statements. Thus, this change is known to be incompatible with statement based replication, and may break some applications or user-generated test suites that depend on sequential auto increment. The previous default can be restored by setting `innodb_autoinc_lock_mode=1;`

- The default value of the `innodb_flush_neighbors` system variable changes from `1` (enable) to `0` (disable). This is done because fast IO (SSDs) is now the default for deployment. We expect that for the majority of users, this results in a small performance gain. Users who are using slower hard drives may see a performance loss, and are encouraged to revert to the previous defaults by setting `innodb_flush_neighbors=1`.

- The default value of the `innodb_max_dirty_pages_pct_lwm` system variable changed from `0` (%) to `10` (%). With `innodb_max_dirty_pages_pct_lwm=10`, InnoDB increases its flushing activity when >10% of the buffer pool contains modified (‘dirty’) pages. The purpose of this change is to trade off peak throughput slightly, in exchange for more consistent performance.

- The default value of the `innodb_max_dirty_pages_pct` system variable changed from `75` (%) to `90` (%). This change combines with the change to `innodb_max_dirty_pages_pct_lwm` and together they ensure a smooth InnoDB flushing behavior, avoiding flushing bursts. To revert to the previous behavior, set `innodb_max_dirty_pages_pct=75` and `innodb_max_dirty_pages_pct_lwm=0`.

Performance Schema Defaults

- Performance Schema Meta Data Locking (MDL) instrumentation is turned on by default. The compiled default for `performance-schema-instrument='wait/lock/metadata/sql/%=ON'` changed from `OFF` to `ON`. This is an enabler for adding MDL oriented views in SYS.

- Performance Schema Memory instrumentation is turned on by default. The compiled default for `performance-schema-instrument='memory/%=COUNTED'` changed from `OFF` to `COUNTED`. This is important because the accounting is incorrect if instrumentation is enabled after server start, and you could get a negative balance from missing an allocation, but catching a free.

- Performance Schema Transaction instrumentation is turned on by default. The compiled default for `performance-schema-consumer-events-transactions-current=ON`, `performance-schema-consumer-events-transactions-history=ON`, and `performance-schema-instrument='transaction%=ON'` changed from `OFF` to `ON`.

Replication Defaults

- The default value of the `log_bin` system variable changed from `OFF` to `ON`. In other words, binary logging is enabled by default. Nearly all production installations have the binary log enabled as it is used for replication and point-in-time recovery. Thus, by enabling binary log by default we eliminate one configuration step, enabling it later requires a `mysqld` restart. Enabling it by default also provides better test coverage and it becomes easier to spot performance regressions. Remember to also set `server_id` (see following change). The 8.0 default behavior is as if you issued `./mysqld --log-bin --server-id=1`. If you are on 8.0 and want 5.7 behavior you can issue `./mysqld --skip-log-bin --server-id=0`.

- The default value of the `server_id` system variable changed from `0` to `1` (combines with the change to `log_bin=ON`). The server can be started with this default ID, but in practice you must set the `server-id` according to the replication infrastructure being deployed, to avoid having duplicate server ids.
- The default value of the `log-slave-updates` system variable changed from `OFF` to `ON`. This causes a replica to log replicated events into its own binary log. This option is required for Group Replication, and also ensures correct behavior in various replication chain setups, which have become the norm today.
- The default value of the `expire_logs_days` system variable changed from `0` to `30`. The new default `30` causes `mysqld` to periodically purge unused binary logs that are older than 30 days. This change helps prevent excessive amounts of disk space being wasted on binary logs that are no longer needed for replication or recovery purposes. The old value of `0` disables any automatic binary log purges.
- The default value of the `master_info_repository` and `relay_log_info_repository` system variables change from `FILE` to `TABLE`. Thus in 8.0, replication metadata is stored in InnoDB by default. This increases reliability to try and achieve crash safe replication by default.
- The default value of the `transaction-write-set-extraction` system variable changed from `OFF` to `XXHASH64`. This change enables transaction write sets by default. By using Transaction Write Sets, the source has to do slightly more work to generate the write sets, but the result is helpful in conflict detection. This is a requirement for Group Replication and the new default makes it easy to enable binary log writeset parallelization on the source to speed up replication.
- The default value of the `slave_rows_search_algorithms` system variable changed from `INDEX_SCAN,TABLE_SCAN` to `INDEX_SCAN,HASH_SCAN`. This change speeds up row-based replication by reducing the number of table scans the replica applier has to do to apply the changes to a table without a primary key.
- The default value of the `slave_pending_jobs_size_max` system variable changed from `16M` to `128M`. This change increases the amount of memory available to multithreaded replicas.
- The default value of the `gtid_executed_compression_period` system variable changed from `1000` to `0`. This change ensures that compression of the `mysql.gtid_executed` table only occurs implicitly as required.

Group Replication Defaults

- The default value of `group_replication_autorejoin_tries` changed from 0 to 3, which means that automatic rejoin is enabled by default. This system variable specifies the number of tries that a member makes to automatically rejoin the group if it is expelled, or if it is unable to contact a majority of the group before the `group_replication_unreachable_majority_timeout` setting is reached.
- The default value of `group_replication_exit_state_action` changed from `ABORT_SERVER` to `READ_ONLY`. This means that when a member exits the group, for example after a network failure, the instance becomes read-only, rather than being shut down.
- The default value of `group_replication_member_expel_timeout` changed from 0 to 5, meaning that a member suspected of having lost contact with the group is liable for expulsion 5 seconds after the 5-second detection period.

Most of these defaults are reasonably good for both development and production environments. There is one exception to this, we decided to keep the new option called `innodb_dedicated_server` set to `OFF` although we recommend it to be `ON` for production environments. The reason for defaulting to `OFF` is that it causes shared environments such as developer laptops to become unusable, because it takes *all* the memory it can find.

For production environments we recommend setting `innodb_dedicated_server` to `ON`. When set to `ON` the following InnoDB variables (if not specified explicitly) are autoscaled based on the available

memory `innodb_buffer_pool_size`, `innodb_log_file_size`, and `innodb_flush_method`. See Section 15.8.12, “Enabling Automatic Configuration for a Dedicated MySQL Server”.

Although the new defaults are the best configuration choices for most use cases, there are special cases, as well as legacy reasons for using existing 5.7 configuration choices. For example, some people prefer to upgrade to 8.0 with as few changes to their applications or operational environment as possible. We recommend to evaluate all the new defaults and use as many as you can. Most new defaults can be tested in 5.7, so you can validate the new defaults in 5.7 production before upgrading to 8.0. For the few defaults where you need your old 5.7 value, set the corresponding configuration variable or startup option in your operational environment.

MySQL 8.0 has the Performance Schema `variables_info` table, which shows for each system variable the source from which it was most recently set, as well as its range of values. This provides SQL access to all there is to know about a configuration variable and its values.

2.10.5 Preparing Your Installation for Upgrade

Before upgrading to the latest MySQL 8.0 release, ensure the upgrade readiness of your current MySQL 5.7 or MySQL 8.0 server instance by performing the preliminary checks described below. The upgrade process may fail otherwise.

Tip

Consider using the MySQL Shell upgrade checker utility that enables you to verify whether MySQL server instances are ready for upgrade. You can select a target MySQL Server release to which you plan to upgrade, ranging from the MySQL Server 8.0.11 up to the MySQL Server release number that matches the current MySQL Shell release number. The upgrade checker utility carries out the automated checks that are relevant for the specified target release, and advises you of further relevant checks that you should make manually. The upgrade checker works for all GA releases of MySQL 5.7 and 8.0. Installation instructions for MySQL Shell can be found here.

Preliminary checks:

1. The following issues must not be present:

   - There must be no tables that use obsolete data types or functions.

     In-place upgrade to MySQL 8.0 is not supported if tables contain old temporal columns in pre-5.6.4 format (`TIME`, `DATETIME`, and `TIMESTAMP` columns without support for fractional seconds precision). If your tables still use the old temporal column format, upgrade them using `REPAIR TABLE` before attempting an in-place upgrade to MySQL 8.0. For more information, see Server Changes, in MySQL 5.7 Reference Manual.

   - There must be no orphan `.frm` files.

   - Triggers must not have a missing or empty definer or an invalid creation context (indicated by the `character_set_client`, `collation_connection`, `Database Collation` attributes displayed by `SHOW TRIGGERS` or the `INFORMATION_SCHEMA TRIGGERS` table). Any such triggers must be dumped and restored to fix the issue.

   To check for these issues, execute this command:

   ```
   mysqlcheck -u root -p --all-databases --check-upgrade
   ```

   If `mysqlcheck` reports any errors, correct the issues.

2. There must be no partitioned tables that use a storage engine that does not have native partitioning support. To identify such tables, execute this query:

   ```
   SELECT TABLE_SCHEMA, TABLE_NAME
   ```

```
FROM INFORMATION_SCHEMA.TABLES
WHERE ENGINE NOT IN ('innodb', 'ndbcluster')
AND CREATE_OPTIONS LIKE '%partitioned%';
```

Any table reported by the query must be altered to use `InnoDB` or be made nonpartitioned. To change a table storage engine to `InnoDB`, execute this statement:

```
ALTER TABLE table_name ENGINE = INNODB;
```

For information about converting `MyISAM` tables to `InnoDB`, see Section 15.6.1.5, “Converting Tables from MyISAM to InnoDB”.

To make a partitioned table nonpartitioned, execute this statement:

```
ALTER TABLE table_name REMOVE PARTITIONING;
```

3. Some keywords may be reserved in MySQL 8.0 that were not reserved previously. See Section 9.3, “Keywords and Reserved Words”. This can cause words previously used as identifiers to become illegal. To fix affected statements, use identifier quoting. See Section 9.2, “Schema Object Names”.

4. There must be no tables in the MySQL 5.7 `mysql` system database that have the same name as a table used by the MySQL 8.0 data dictionary. To identify tables with those names, execute this query:

```
SELECT TABLE_SCHEMA, TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
WHERE LOWER(TABLE_SCHEMA) = 'mysql'
and LOWER(TABLE_NAME) IN
(
'catalogs',
'character_sets',
'check_constraints',
'collations',
'column_statistics',
'column_type_elements',
'columns',
'dd_properties',
'events',
'foreign_key_column_usage',
'foreign_keys',
'index_column_usage',
'index_partitions',
'index_stats',
'indexes',
'parameter_type_elements',
'parameters',
'resource_groups',
'routines',
'schemata',
'st_spatial_reference_systems',
'table_partition_values',
'table_partitions',
'table_stats',
'tables',
'tablespace_files',
'tablespaces',
'triggers',
'view_routine_usage',
'view_table_usage'
);
```

Any tables reported by the query must be dropped or renamed (use `RENAME TABLE`). This may also entail changes to applications that use the affected tables.

5. There must be no tables that have foreign key constraint names longer than 64 characters. Use this query to identify tables with constraint names that are too long:

```
SELECT TABLE_SCHEMA, TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
```

```
WHERE TABLE_NAME IN
  (SELECT LEFT(SUBSTR(ID,INSTR(ID,'/')+1),
              INSTR(SUBSTR(ID,INSTR(ID,'/')+1),'_ibfk_')-1)
   FROM INFORMATION_SCHEMA.INNODB_SYS_FOREIGN
   WHERE LENGTH(SUBSTR(ID,INSTR(ID,'/')+1))>64);
```

For a table with a constraint name that exceeds 64 characters, drop the constraint and add it back with constraint name that does not exceed 64 characters (use `ALTER TABLE`).

6. There must be no obsolete SQL modes defined by `sql_mode` system variable. Attempting to use an obsolete SQL mode prevents MySQL 8.0 from starting. Applications that use obsolete SQL modes should be revised to avoid them. For information about SQL modes removed in MySQL 8.0, see Server Changes.

7. There must be no views with explicitly defined columns names that exceed 64 characters (views with column names up to 255 characters were permitted in MySQL 5.7). To avoid upgrade errors, such views should be altered before upgrading. Currently, the only method of identify views with column names that exceed 64 characters is to inspect the view definition using `SHOW CREATE VIEW`. You can also inspect view definitions by querying the Information Schema `VIEWS` table.

8. There must be no tables or stored procedures with individual `ENUM` or `SET` column elements that exceed 255 characters or 1020 bytes in length. Prior to MySQL 8.0, the maximum combined length of `ENUM` or `SET` column elements was 64K. In MySQL 8.0, the maximum character length of an individual `ENUM` or `SET` column element is 255 characters, and the maximum byte length is 1020 bytes. (The 1020 byte limit supports multibyte character sets). Before upgrading to MySQL 8.0, modify any `ENUM` or `SET` column elements that exceed the new limits. Failing to do so causes the upgrade to fail with an error.

9. Before upgrading to MySQL 8.0.13 or higher, there must be no table partitions that reside in shared `InnoDB` tablespaces, which include the system tablespace and general tablespaces. Identify table partitions in shared tablespaces by querying `INFORMATION_SCHEMA`:

   If upgrading from MySQL 5.7, run this query:

   ```
   SELECT DISTINCT NAME, SPACE, SPACE_TYPE FROM INFORMATION_SCHEMA.INNODB_SYS_TABLES
     WHERE NAME LIKE '%#P#%' AND SPACE_TYPE NOT LIKE 'Single';
   ```

   If upgrading from an earlier MySQL 8.0 release, run this query:

   ```
   SELECT DISTINCT NAME, SPACE, SPACE_TYPE FROM INFORMATION_SCHEMA.INNODB_TABLES
     WHERE NAME LIKE '%#P#%' AND SPACE_TYPE NOT LIKE 'Single';
   ```

   Move table partitions from shared tablespaces to file-per-table tablespaces using `ALTER TABLE ... REORGANIZE PARTITION`:

   ```
   ALTER TABLE table_name REORGANIZE PARTITION partition_name
     INTO (partition_definition TABLESPACE=innodb_file_per_table);
   ```

10. There must be no queries and stored program definitions from MySQL 8.0.12 or lower that use `ASC` or `DESC` qualifiers for `GROUP BY` clauses. Otherwise, upgrading to MySQL 8.0.13 or higher may fail, as may replicating to MySQL 8.0.13 or higher replica servers. For additional details, see SQL Changes.

11. Your MySQL 5.7 installation must not use features that are not supported by MySQL 8.0. Any changes here are necessarily installation specific, but the following example illustrates the kind of thing to look for:

    Some server startup options and system variables have been removed in MySQL 8.0. See Features Removed in MySQL 8.0, and Section 1.4, “Server and Status Variables and Options

Added, Deprecated, or Removed in MySQL 8.0". If you use any of these, an upgrade requires configuration changes.

Example: Because the data dictionary provides information about database objects, the server no longer checks directory names in the data directory to find databases. Consequently, the `--ignore-db-dir` option is extraneous and has been removed. To handle this, remove any instances of `--ignore-db-dir` from your startup configuration. In addition, remove or move the named data directory subdirectories before upgrading to MySQL 8.0. (Alternatively, let the 8.0 server add those directories to the data dictionary as databases, then remove each of those databases using `DROP DATABASE`.)

12. If you intend to change the `lower_case_table_names` setting to 1 at upgrade time, ensure that schema and table names are lowercase before upgrading. Otherwise, a failure could occur due to a schema or table name lettercase mismatch. You can use the following queries to check for schema and table names containing uppercase characters:

```
mysql> select TABLE_NAME, if(sha(TABLE_NAME) !=sha(lower(TABLE_NAME)),'Yes','No') as UpperCase from
```

As of MySQL 8.0.19, if `lower_case_table_names=1`, table and schema names are checked by the upgrade process to ensure that all characters are lowercase. If table or schema names are found to contain uppercase characters, the upgrade process fails with an error.

Note

Changing the `lower_case_table_names` setting at upgrade time is not recommended.

If upgrade to MySQL 8.0 fails due to any of the issues outlined above, the server reverts all changes to the data directory. In this case, remove all redo log files and restart the MySQL 5.7 server on the existing data directory to address the errors. The redo log files (`ib_logfile*`) reside in the MySQL data directory by default. After the errors are fixed, perform a slow shutdown (by setting `innodb_fast_shutdown=0`) before attempting the upgrade again.

2.10.6 Upgrading MySQL Binary or Package-based Installations on Unix/Linux

This section describes how to upgrade MySQL binary and package-based installations on Unix/Linux. In-place and logical upgrade methods are described.

- In-Place Upgrade
- Logical Upgrade
- MySQL Cluster Upgrade

In-Place Upgrade

An in-place upgrade involves shutting down the old MySQL server, replacing the old MySQL binaries or packages with the new ones, restarting MySQL on the existing data directory, and upgrading any remaining parts of the existing installation that require upgrading. For details about what may need upgrading, see Section 2.10.3, "What the MySQL Upgrade Process Upgrades".

Note

If you are upgrading an installation originally produced by installing multiple RPM packages, upgrade all the packages, not just some. For example, if you previously installed the server and client RPMs, do not upgrade just the server RPM.

For some Linux platforms, MySQL installation from RPM or Debian packages includes systemd support for managing MySQL server startup and shutdown.

> On these platforms, `mysqld_safe` is not installed. In such cases, use systemd for server startup and shutdown instead of the methods used in the following instructions. See Section 2.5.9, “Managing MySQL Server with systemd”.
>
> For upgrades to MySQL Cluster installations, see also MySQL Cluster Upgrade.

To perform an in-place upgrade:

1. Review the information in Section 2.10.1, “Before You Begin”.
2. Ensure the upgrade readiness of your installation by completing the preliminary checks in Section 2.10.5, “Preparing Your Installation for Upgrade”.
3. If you use XA transactions with `InnoDB`, run `XA RECOVER` before upgrading to check for uncommitted XA transactions. If results are returned, either commit or rollback the XA transactions by issuing an `XA COMMIT` or `XA ROLLBACK` statement.
4. If you are upgrading from MySQL 5.7.11 or earlier to MySQL 8.0, and there are encrypted `InnoDB` tablespaces, rotate the keyring master key by executing this statement:

   ```
   ALTER INSTANCE ROTATE INNODB MASTER KEY;
   ```

5. If you normally run your MySQL server configured with `innodb_fast_shutdown` set to `2` (cold shutdown), configure it to perform a fast or slow shutdown by executing either of these statements:

   ```
   SET GLOBAL innodb_fast_shutdown = 1; -- fast shutdown
   SET GLOBAL innodb_fast_shutdown = 0; -- slow shutdown
   ```

   With a fast or slow shutdown, `InnoDB` leaves its undo logs and data files in a state that can be dealt with in case of file format differences between releases.
6. Shut down the old MySQL server. For example:

   ```
   mysqladmin -u root -p shutdown
   ```

7. Upgrade the MySQL binaries or packages. If upgrading a binary installation, unpack the new MySQL binary distribution package. See Obtain and Unpack the Distribution. For package-based installations, install the new packages.
8. Start the MySQL 8.0 server, using the existing data directory. For example:

   ```
   mysqld_safe --user=mysql --datadir=/path/to/existing-datadir &
   ```

   If there are encrypted `InnoDB` tablespaces, use the `--early-plugin-load` option to load the keyring plugin.

   When you start the MySQL 8.0 server, it automatically detects whether data dictionary tables are present. If not, the server creates them in the data directory, populates them with metadata, and then proceeds with its normal startup sequence. During this process, the server upgrades metadata for all database objects, including databases, tablespaces, system and user tables, views, and stored programs (stored procedures and functions, triggers, and Event Scheduler events). The server also removes files that previously were used for metadata storage. For example, after upgrading from MySQL 5.7 to MySQL 8.0, you may notice that tables no longer have `.frm` files.

   If this step fails, the server reverts all changes to the data directory. In this case, you should remove all redo log files, start your MySQL 5.7 server on the same data directory, and fix the cause of any errors. Then perform another slow shutdown of the 5.7 server and start the MySQL 8.0 server to try again.
9. In the previous step, the server upgrades the data dictionary as necessary. Now it is necessary to perform any remaining upgrade operations:

   - As of MySQL 8.0.16, the server does so as part of the previous step, making any changes required in the `mysql` system database between MySQL 5.7 and MySQL 8.0, so that you

can take advantage of new privileges or capabilities. It also brings the Performance Schema, `INFORMATION_SCHEMA`, and `sys` databases up to date for MySQL 8.0, and examines all user databases for incompatibilities with the current version of MySQL.

- Prior to MySQL 8.0.16, the server upgrades only the data dictionary in the previous step. After the MySQL 8.0 server starts successfully, execute `mysql_upgrade` to perform the remaining upgrade tasks:

  ```
  mysql_upgrade -u root -p
  ```

  Then shut down and restart the MySQL server to ensure that any changes made to the system tables take effect. For example:

  ```
  mysqladmin -u root -p shutdown
  mysqld_safe --user=mysql --datadir=/path/to/existing-datadir &
  ```

  The first time you start the MySQL 8.0 server (in an earlier step), you may notice messages in the error log regarding nonupgraded tables. If `mysql_upgrade` has been run successfully, there should be no such messages the second time you start the server.

Note

The upgrade process does not upgrade the contents of the time zone tables. For upgrade instructions, see Section 5.1.15, “MySQL Server Time Zone Support”.

If the upgrade process uses `mysql_upgrade` (that is, prior to MySQL 8.0.16), the process does not upgrade the contents of the help tables, either. For upgrade instructions in that case, see Section 5.1.17, “Server-Side Help Support”.

Logical Upgrade

A logical upgrade involves exporting SQL from the old MySQL instance using a backup or export utility such as `mysqldump` or `mysqlpump`, installing the new MySQL server, and applying the SQL to your new MySQL instance. For details about what may need upgrading, see Section 2.10.3, “What the MySQL Upgrade Process Upgrades”.

Note

For some Linux platforms, MySQL installation from RPM or Debian packages includes systemd support for managing MySQL server startup and shutdown. On these platforms, `mysqld_safe` is not installed. In such cases, use systemd for server startup and shutdown instead of the methods used in the following instructions. See Section 2.5.9, “Managing MySQL Server with systemd”.

Warning

Applying SQL extracted from a previous MySQL release to a new MySQL release may result in errors due to incompatibilities introduced by new, changed, deprecated, or removed features and capabilities. Consequently, SQL extracted from a previous MySQL release may require modification to enable a logical upgrade.

To identify incompatibilities before upgrading to the latest MySQL 8.0 release, perform the steps described in Section 2.10.5, “Preparing Your Installation for Upgrade”.

To perform a logical upgrade:

1. Review the information in Section 2.10.1, “Before You Begin”.

2. Export your existing data from the previous MySQL installation:

```
mysqldump -u root -p
  --add-drop-table --routines --events
  --all-databases --force > data-for-upgrade.sql
```

Note

Use the `--routines` and `--events` options with `mysqldump` (as shown above) if your databases include stored programs. The `--all-databases` option includes all databases in the dump, including the `mysql` database that holds the system tables.

Important

If you have tables that contain generated columns, use the `mysqldump` utility provided with MySQL 5.7.9 or higher to create your dump files. The `mysqldump` utility provided in earlier releases uses incorrect syntax for generated column definitions (Bug #20769542). You can use the Information Schema `COLUMNS` table to identify tables with generated columns.

3. Shut down the old MySQL server. For example:

```
mysqladmin -u root -p shutdown
```

4. Install MySQL 8.0. For installation instructions, see Chapter 2, *Installing and Upgrading MySQL*.

5. Initialize a new data directory, as described in Section 2.9.1, “Initializing the Data Directory”. For example:

```
mysqld --initialize --datadir=/path/to/8.0-datadir
```

Copy the temporary `'root'@'localhost'` password displayed to your screen or written to your error log for later use.

6. Start the MySQL 8.0 server, using the new data directory. For example:

```
mysqld_safe --user=mysql --datadir=/path/to/8.0-datadir &
```

7. Reset the `root` password:

```
$> mysql -u root -p
Enter password: ****  <- enter temporary root password
```

```
mysql> ALTER USER USER() IDENTIFIED BY 'your new password';
```

8. Load the previously created dump file into the new MySQL server. For example:

```
mysql -u root -p --force < data-for-upgrade.sql
```

Note

It is not recommended to load a dump file when GTIDs are enabled on the server (`gtid_mode=ON`), if your dump file includes system tables. `mysqldump` issues DML instructions for the system tables which use the non-transactional MyISAM storage engine, and this combination is not permitted when GTIDs are enabled. Also be aware that loading a dump file from a server with GTIDs enabled, into another server with GTIDs enabled, causes different transaction identifiers to be generated.

9. Perform any remaining upgrade operations:

   - In MySQL 8.0.16 and higher, shut down the server, then restart it with the `--upgrade=FORCE` option to perform the remaining upgrade tasks:

     ```
     mysqladmin -u root -p shutdown
     mysqld_safe --user=mysql --datadir=/path/to/8.0-datadir --upgrade=FORCE &
     ```

     Upon restart with `--upgrade=FORCE`, the server makes any changes required in the `mysql` system schema between MySQL 5.7 and MySQL 8.0, so that you can take advantage of new privileges or capabilities. It also brings the Performance Schema, `INFORMATION_SCHEMA`, and `sys` schema up to date for MySQL 8.0, and examines all user schemas for incompatibilities with the current version of MySQL.

   - Prior to MySQL 8.0.16, execute `mysql_upgrade` to perform the remaining upgrade tasks:

     ```
     mysql_upgrade -u root -p
     ```

     Then shut down and restart the MySQL server to ensure that any changes made to the system tables take effect. For example:

     ```
     mysqladmin -u root -p shutdown
     mysqld_safe --user=mysql --datadir=/path/to/8.0-datadir &
     ```

Note

The upgrade process does not upgrade the contents of the time zone tables. For upgrade instructions, see Section 5.1.15, “MySQL Server Time Zone Support”.

If the upgrade process uses `mysql_upgrade` (that is, prior to MySQL 8.0.16), the process does not upgrade the contents of the help tables, either. For upgrade instructions in that case, see Section 5.1.17, “Server-Side Help Support”.

Note

Loading a dump file that contains a MySQL 5.7 `mysql` schema re-creates two tables that are no longer used: `event` and `proc`. (The corresponding MySQL 8.0 tables are `events` and `routines`, both of which are data dictionary tables and are protected.) After you are satisfied that the upgrade was successful, you can remove the `event` and `proc` tables by executing these SQL statements:

```
DROP TABLE mysql.event;
DROP TABLE mysql.proc;
```

MySQL Cluster Upgrade

The information in this section is an adjunct to the in-place upgrade procedure described in In-Place Upgrade, for use if you are upgrading MySQL Cluster.

As of MySQL 8.0.16, a MySQL Cluster upgrade can be performed as a regular rolling upgrade, following the usual three ordered steps:

1. Upgrade MGM nodes.
2. Upgrade data nodes one at a time.
3. Upgrade API nodes one at a time (including MySQL servers).

The way to upgrade each of the nodes remains almost the same as prior to MySQL 8.0.16 because there is a separation between upgrading the data dictionary and upgrading the system tables. There are two steps to upgrading each individual `mysqld`:

1. Import the data dictionary.

   Start the new server with the `--upgrade=MINIMAL` option to upgrade the data dictionary but not the system tables. This is essentially the same as the pre-MySQL 8.0.16 action of starting the server and not invoking `mysql_upgrade`.

   The MySQL server must be connected to `NDB` for this phase to complete. If any `NDB` or `NDBINFO` tables exist, and the server cannot connect to the cluster, it exits with an error message:

   ```
   Failed to Populate DD tables.
   ```

2. Upgrade the system tables.

   Prior to MySQL 8.0.16, the DBA invokes the `mysql_upgrade` client to upgrade the system tables. As of MySQL 8.0.16, the server performs this action: To upgrade the system tables, restart each individual `mysqld` without the `--upgrade=MINIMAL` option.

2.10.7 Upgrading MySQL with the MySQL Yum Repository

For supported Yum-based platforms (see Section 2.5.1, “Installing MySQL on Linux Using the MySQL Yum Repository”, for a list), you can perform an in-place upgrade for MySQL (that is, replacing the old version and then running the new version using the old data files) with the MySQL Yum repository.

Notes

- Before performing any update to MySQL, follow carefully the instructions in Section 2.10, “Upgrading MySQL”. Among other instructions discussed there, it is especially important to back up your database before the update.
- The following instructions assume you have installed MySQL with the MySQL Yum repository or with an RPM package directly downloaded from MySQL Developer Zone's MySQL Download page; if that is not the case, following the instructions in Replacing a Third-Party Distribution of MySQL Using the MySQL Yum Repository.

Selecting a Target Series

1. By default, the MySQL Yum repository updates MySQL to the latest version in the release series you have chosen during installation (see Selecting a Release Series for details), which means, for example, a 5.7.x installation is *not* updated to a 8.0.x release automatically. To update to another release series, you must first disable the subrepository for the series that has been selected (by default, or by yourself) and enable the subrepository for your target series. To do that, see the general instructions given in Selecting a Release Series. For upgrading from MySQL 5.7 to 8.0, perform the *reverse* of the steps illustrated in Selecting a Release Series, disabling the subrepository for the MySQL 5.7 series and enabling that for the MySQL 8.0 series.

   As a general rule, to upgrade from one release series to another, go to the next series rather than skipping a series. For example, if you are currently running MySQL 5.6 and wish to upgrade to 8.0, upgrade to MySQL 5.7 first before upgrading to 8.0.

Important

For important information about upgrading from MySQL 5.7 to 8.0, see Upgrading from MySQL 5.7 to 8.0.

Upgrading MySQL

2. Upgrade MySQL and its components by the following command, for platforms that are not dnf-enabled:

   ```
   sudo yum update mysql-server
   ```

For platforms that are dnf-enabled:

```
sudo dnf upgrade mysql-server
```

Alternatively, you can update MySQL by telling Yum to update everything on your system, which might take considerably more time. For platforms that are not dnf-enabled:

```
sudo yum update
```

For platforms that are dnf-enabled:

```
sudo dnf upgrade
```

Restarting MySQL

The MySQL server always restarts after an update by Yum. Prior to MySQL 8.0.16, run `mysql_upgrade` after the server restarts to check and possibly resolve any incompatibilities between the old data and the upgraded software. `mysql_upgrade` also performs other functions; for details, see Section 4.4.5, “mysql_upgrade — Check and Upgrade MySQL Tables”. As of MySQL 8.0.16, this step is not required, as the server performs all tasks previously handled by `mysql_upgrade`.

You can also update only a specific component. Use the following command to list all the installed packages for the MySQL components (for dnf-enabled systems, replace `yum` in the command with `dnf`):

```
sudo yum list installed | grep "^mysql"
```

After identifying the package name of the component of your choice, update the package with the following command, replacing *`package-name`* with the name of the package. For platforms that are not dnf-enabled:

```
sudo yum update package-name
```

For dnf-enabled platforms:

```
sudo dnf upgrade package-name
```

Upgrading the Shared Client Libraries

After updating MySQL using the Yum repository, applications compiled with older versions of the shared client libraries should continue to work.

If you recompile applications and dynamically link them with the updated libraries: As typical with new versions of shared libraries where there are differences or additions in symbol versioning between the newer and older libraries (for example, between the newer, standard 8.0 shared client libraries and some older—prior or variant—versions of the shared libraries shipped natively by the Linux distributions' software repositories, or from some other sources), any applications compiled using the updated, newer shared libraries require those updated libraries on systems where the applications are deployed. As expected, if those libraries are not in place, the applications requiring the shared libraries fail. For this reason, be sure to deploy the packages for the shared libraries from MySQL on those systems. To do this, add the MySQL Yum repository to the systems (see Adding the MySQL Yum Repository) and install the latest shared libraries using the instructions given in Installing Additional MySQL Products and Components with Yum.

2.10.8 Upgrading MySQL with the MySQL APT Repository

On Debian and Ubuntu platforms, to perform an in-place upgrade of MySQL and its components, use the MySQL APT repository. See Upgrading MySQL with the MySQL APT Repository in A Quick Guide to Using the MySQL APT Repository.

2.10.9 Upgrading MySQL with the MySQL SLES Repository

On the SUSE Linux Enterprise Server (SLES) platform, to perform an in-place upgrade of MySQL and its components, use the MySQL SLES repository. See Upgrading MySQL with the MySQL SLES Repository in A Quick Guide to Using the MySQL SLES Repository.

2.10.10 Upgrading MySQL on Windows

There are two approaches for upgrading MySQL on Windows:

- Using MySQL Installer
- Using the Windows ZIP archive distribution

The approach you select depends on how the existing installation was performed. Before proceeding, review Section 2.10, “Upgrading MySQL” for additional information on upgrading MySQL that is not specific to Windows.

Note

Whichever approach you choose, always back up your current MySQL installation before performing an upgrade. See Section 7.2, “Database Backup Methods”.

Upgrades between non-GA releases (or from a non-GA release to a GA release) are not supported. Significant development changes take place in non-GA releases and you may encounter compatibility issues or problems starting the server.

Note

MySQL Installer does not support upgrades between *Community* releases and *Commercial* releases. If you require this type of upgrade, perform it using the ZIP archive approach.

Upgrading MySQL with MySQL Installer

Performing an upgrade with MySQL Installer is the best approach when the current server installation was performed with it and the upgrade is within the current release series. MySQL Installer does not support upgrades between release series, such as from 5.7 to 8.0, and it does not provide an upgrade indicator to prompt you to upgrade. For instructions on upgrading between release series, see Upgrading MySQL Using the Windows ZIP Distribution.

To perform an upgrade using MySQL Installer:

1. Start MySQL Installer.
2. From the dashboard, click **Catalog** to download the latest changes to the catalog. The installed server can be upgraded only if the dashboard displays an arrow next to the version number of the server.
3. Click **Upgrade**. All products that have a newer version now appear in a list.

   Note

   MySQL Installer deselects the server upgrade option for milestone releases (Pre-Release) in the same release series. In addition, it displays a warning to indicate that the upgrade is not supported, identifies the risks of continuing, and provides a summary of the steps to perform an upgrade manually. You can reselect server upgrade and proceed at your own risk.
4. Deselect all but the MySQL server product, unless you intend to upgrade other products at this time, and click **Next**.

5. Click **Execute** to start the download. When the download finishes, click **Next** to begin the upgrade operation.

   Upgrades to MySQL 8.0.16 and higher may show an option to skip the upgrade check and process for system tables. For more information about this option, see Important server upgrade conditions.

6. Configure the server.

Upgrading MySQL Using the Windows ZIP Distribution

To perform an upgrade using the Windows ZIP archive distribution:

1. Download the latest Windows ZIP Archive distribution of MySQL from https://dev.mysql.com/downloads/.

2. If the server is running, stop it. If the server is installed as a service, stop the service with the following command from the command prompt:

   ```
   C:\> SC STOP mysqld_service_name
   ```

   Alternatively, use `NET STOP mysqld_service_name` .

   If you are not running the MySQL server as a service, use `mysqladmin` to stop it. For example, before upgrading from MySQL 5.7 to 8.0, use `mysqladmin` from MySQL 5.7 as follows:

   ```
   C:\> "C:\Program Files\MySQL\MySQL Server 5.7\bin\mysqladmin" -u root shutdown
   ```

   Note

   If the MySQL `root` user account has a password, invoke `mysqladmin` with the `-p` option and enter the password when prompted.

3. Extract the ZIP archive. You may either overwrite your existing MySQL installation (usually located at `C:\mysql`), or install it into a different directory, such as `C:\mysql8`. Overwriting the existing installation is recommended.

4. Restart the server. For example, use the `SC START mysqld_service_name` or `NET START mysqld_service_name` command if you run MySQL as a service, or invoke `mysqld` directly otherwise.

5. Prior to MySQL 8.0.16, run `mysql_upgrade` as Administrator to check your tables, attempt to repair them if necessary, and update your grant tables if they have changed so that you can take advantage of any new capabilities. See Section 4.4.5, “mysql_upgrade — Check and Upgrade MySQL Tables”. As of MySQL 8.0.16, this step is not required, as the server performs all tasks previously handled by `mysql_upgrade`.

6. If you encounter errors, see Section 2.3.5, “Troubleshooting a Microsoft Windows MySQL Server Installation”.

2.10.11 Upgrading a Docker Installation of MySQL

To upgrade a Docker installation of MySQL, refer to Upgrading a MySQL Server Container.

2.10.12 Upgrade Troubleshooting

- A schema mismatch in a MySQL 5.7 instance between the `.frm` file of a table and the `InnoDB` data dictionary can cause an upgrade to MySQL 8.0 to fail. Such mismatches may be due to `.frm` file corruption. To address this issue, dump and restore affected tables before attempting the upgrade again.

- If problems occur, such as that the new `mysqld` server does not start, verify that you do not have an old `my.cnf` file from your previous installation. You can check this with the `--print-defaults` option (for example, `mysqld --print-defaults`). If this command displays anything other than the program name, you have an active `my.cnf` file that affects server or client operation.

- If, after an upgrade, you experience problems with compiled client programs, such as `Commands out of sync` or unexpected core dumps, you probably have used old header or library files when compiling your programs. In this case, check the date for your `mysql.h` file and `libmysqlclient.a` library to verify that they are from the new MySQL distribution. If not, recompile your programs with the new headers and libraries. Recompilation might also be necessary for programs compiled against the shared client library if the library major version number has changed (for example, from `libmysqlclient.so.20` to `libmysqlclient.so.21`).

- If you have created a loadable function with a given name and upgrade MySQL to a version that implements a new built-in function with the same name, the loadable function becomes inaccessible. To correct this, use `DROP FUNCTION` to drop the loadable function, and then use `CREATE FUNCTION` to re-create the loadable function with a different nonconflicting name. The same is true if the new version of MySQL implements a built-in function with the same name as an existing stored function. See Section 9.2.5, “Function Name Parsing and Resolution”, for the rules describing how the server interprets references to different kinds of functions.

- If upgrade to MySQL 8.0 fails due to any of the issues outlined in Section 2.10.5, “Preparing Your Installation for Upgrade”, the server reverts all changes to the data directory. In this case, remove all redo log files and restart the MySQL 5.7 server on the existing data directory to address the errors. The redo log files (`ib_logfile*`) reside in the MySQL data directory by default. After the errors are fixed, perform a slow shutdown (by setting `innodb_fast_shutdown=0`) before attempting the upgrade again.

2.10.13 Rebuilding or Repairing Tables or Indexes

This section describes how to rebuild or repair tables or indexes, which may be necessitated by:

- Changes to how MySQL handles data types or character sets. For example, an error in a collation might have been corrected, necessitating a table rebuild to update the indexes for character columns that use the collation.

- Required table repairs or upgrades reported by `CHECK TABLE`, `mysqlcheck`, or `mysql_upgrade`.

Methods for rebuilding a table include:

- Dump and Reload Method
- ALTER TABLE Method
- REPAIR TABLE Method

Dump and Reload Method

If you are rebuilding tables because a different version of MySQL cannot handle them after a binary (in-place) upgrade or downgrade, you must use the dump-and-reload method. Dump the tables *before* upgrading or downgrading using your original version of MySQL. Then reload the tables *after* upgrading or downgrading.

If you use the dump-and-reload method of rebuilding tables only for the purpose of rebuilding indexes, you can perform the dump either before or after upgrading or downgrading. Reloading still must be done afterward.

If you need to rebuild an `InnoDB` table because a `CHECK TABLE` operation indicates that a table upgrade is required, use `mysqldump` to create a dump file and `mysql` to reload the file. If the `CHECK TABLE` operation indicates that there is a corruption or causes `InnoDB` to fail, refer to Section 15.21.3,

"Forcing InnoDB Recovery" for information about using the `innodb_force_recovery` option to restart `InnoDB`. To understand the type of problem that `CHECK TABLE` may be encountering, refer to the `InnoDB` notes in Section 13.7.3.2, "CHECK TABLE Statement".

To rebuild a table by dumping and reloading it, use `mysqldump` to create a dump file and `mysql` to reload the file:

```
mysqldump db_name t1 > dump.sql
mysql db_name < dump.sql
```

To rebuild all the tables in a single database, specify the database name without any following table name:

```
mysqldump db_name > dump.sql
mysql db_name < dump.sql
```

To rebuild all tables in all databases, use the `--all-databases` option:

```
mysqldump --all-databases > dump.sql
mysql < dump.sql
```

ALTER TABLE Method

To rebuild a table with `ALTER TABLE`, use a "null" alteration; that is, an `ALTER TABLE` statement that "changes" the table to use the storage engine that it already has. For example, if `t1` is an `InnoDB` table, use this statement:

```
ALTER TABLE t1 ENGINE = InnoDB;
```

If you are not sure which storage engine to specify in the `ALTER TABLE` statement, use `SHOW CREATE TABLE` to display the table definition.

REPAIR TABLE Method

The `REPAIR TABLE` method is only applicable to `MyISAM`, `ARCHIVE`, and `CSV` tables.

You can use `REPAIR TABLE` if the table checking operation indicates that there is a corruption or that an upgrade is required. For example, to repair a `MyISAM` table, use this statement:

```
REPAIR TABLE t1;
```

`mysqlcheck --repair` provides command-line access to the `REPAIR TABLE` statement. This can be a more convenient means of repairing tables because you can use the `--databases` or `--all-databases` option to repair all tables in specific databases or all databases, respectively:

```
mysqlcheck --repair --databases db_name ...
mysqlcheck --repair --all-databases
```

2.10.14 Copying MySQL Databases to Another Machine

In cases where you need to transfer databases between different architectures, you can use `mysqldump` to create a file containing SQL statements. You can then transfer the file to the other machine and feed it as input to the `mysql` client.

Use `mysqldump --help` to see what options are available.

Note

If GTIDs are in use on the server where you create the dump (`gtid_mode=ON`), by default, `mysqldump` includes the contents of the `gtid_executed` set in the dump to transfer these to the new machine. The results of this can vary depending on the MySQL Server versions involved. Check the description for `mysqldump's --set-gtid-purged` option to find what happens with the

versions you are using, and how to change the behavior if the outcome of the default behavior is not suitable for your situation.

The easiest (although not the fastest) way to move a database between two machines is to run the following commands on the machine on which the database is located:

```
mysqladmin -h 'other_hostname' create db_name
mysqldump db_name | mysql -h 'other_hostname' db_name
```

If you want to copy a database from a remote machine over a slow network, you can use these commands:

```
mysqladmin create db_name
mysqldump -h 'other_hostname' --compress db_name | mysql db_name
```

You can also store the dump in a file, transfer the file to the target machine, and then load the file into the database there. For example, you can dump a database to a compressed file on the source machine like this:

```
mysqldump --quick db_name | gzip > db_name.gz
```

Transfer the file containing the database contents to the target machine and run these commands there:

```
mysqladmin create db_name
gunzip < db_name.gz | mysql db_name
```

You can also use `mysqldump` and `mysqlimport` to transfer the database. For large tables, this is much faster than simply using `mysqldump`. In the following commands, *`DUMPDIR`* represents the full path name of the directory you use to store the output from `mysqldump`.

First, create the directory for the output files and dump the database:

```
mkdir DUMPDIR
mysqldump --tab=DUMPDIR
   db_name
```

Then transfer the files in the *`DUMPDIR`* directory to some corresponding directory on the target machine and load the files into MySQL there:

```
mysqladmin create db_name           # create database
cat DUMPDIR/*.sql | mysql db_name   # create tables in database
mysqlimport db_name
   DUMPDIR/*.txt   # load data into tables
```

Do not forget to copy the `mysql` database because that is where the grant tables are stored. You might have to run commands as the MySQL `root` user on the new machine until you have the `mysql` database in place.

After you import the `mysql` database on the new machine, execute `mysqladmin flush-privileges` so that the server reloads the grant table information.

2.11 Downgrading MySQL

Downgrade from MySQL 8.0 to MySQL 5.7, or from a MySQL 8.0 release to a previous MySQL 8.0 release, is not supported. The only supported alternative is to restore a backup taken *before* upgrading. It is therefore imperative that you back up your data before starting the upgrade process.

2.12 Perl Installation Notes

The Perl DBI module provides a generic interface for database access. You can write a DBI script that works with many different database engines without change. To use DBI, you must install the DBI module, as well as a DataBase Driver (DBD) module for each type of database server you want to access. For MySQL, this driver is the DBD::mysql module.

Note

Perl support is not included with MySQL distributions. You can obtain the necessary modules from http://search.cpan.org for Unix, or by using the ActiveState ppm program on Windows. The following sections describe how to do this.

The DBI/DBD interface requires Perl 5.6.0, and 5.6.1 or later is preferred. DBI *does not work* if you have an older version of Perl. You should use DBD::mysql 4.009 or higher. Although earlier versions are available, they do not support the full functionality of MySQL 8.0.

2.12.1 Installing Perl on Unix

MySQL Perl support requires that you have installed MySQL client programming support (libraries and header files). Most installation methods install the necessary files. If you install MySQL from RPM files on Linux, be sure to install the developer RPM as well. The client programs are in the client RPM, but client programming support is in the developer RPM.

The files you need for Perl support can be obtained from the CPAN (Comprehensive Perl Archive Network) at http://search.cpan.org.

The easiest way to install Perl modules on Unix is to use the CPAN module. For example:

```
$> perl -MCPAN -e shell
cpan> install DBI
cpan> install DBD::mysql
```

The DBD::mysql installation runs a number of tests. These tests attempt to connect to the local MySQL server using the default user name and password. (The default user name is your login name on Unix, and ODBC on Windows. The default password is “no password.”) If you cannot connect to the server with those values (for example, if your account has a password), the tests fail. You can use `force install DBD::mysql` to ignore the failed tests.

DBI requires the Data::Dumper module. It may be installed; if not, you should install it before installing DBI.

It is also possible to download the module distributions in the form of compressed tar archives and build the modules manually. For example, to unpack and build a DBI distribution, use a procedure such as this:

1. Unpack the distribution into the current directory:

   ```
   $> gunzip < DBI-VERSION.tar.gz | tar xvf -
   ```

   This command creates a directory named DBI-*VERSION*.

2. Change location into the top-level directory of the unpacked distribution:

   ```
   $> cd DBI-VERSION
   ```

3. Build the distribution and compile everything:

   ```
   $> perl Makefile.PL
   $> make
   $> make test
   $> make install
   ```

The `make test` command is important because it verifies that the module is working. Note that when you run that command during the `DBD::mysql` installation to exercise the interface code, the MySQL server must be running or the test fails.

It is a good idea to rebuild and reinstall the `DBD::mysql` distribution whenever you install a new release of MySQL. This ensures that the latest versions of the MySQL client libraries are installed correctly.

If you do not have access rights to install Perl modules in the system directory or if you want to install local Perl modules, the following reference may be useful: http://learn.perl.org/faq/perlfaq8.html#How-do-I-keep-my-own-module-library-directory-

2.12.2 Installing ActiveState Perl on Windows

On Windows, you should do the following to install the MySQL `DBD` module with ActiveState Perl:

1. Get ActiveState Perl from http://www.activestate.com/Products/ActivePerl/ and install it.
2. Open a console window.
3. If necessary, set the `HTTP_proxy` variable. For example, you might try a setting like this:

   ```
   C:\> set HTTP_proxy=my.proxy.com:3128
   ```

4. Start the PPM program:

   ```
   C:\> C:\perl\bin\ppm.pl
   ```

5. If you have not previously done so, install `DBI`:

   ```
   ppm> install DBI
   ```

6. If this succeeds, run the following command:

   ```
   ppm> install DBD-mysql
   ```

This procedure should work with ActiveState Perl 5.6 or higher.

If you cannot get the procedure to work, you should install the ODBC driver instead and connect to the MySQL server through ODBC:

```
use DBI;
$dbh= DBI->connect("DBI:ODBC:$dsn",$user,$password) ||
  die "Got error $DBI::errstr when connecting to $dsn\n";
```

2.12.3 Problems Using the Perl DBI/DBD Interface

If Perl reports that it cannot find the `../mysql/mysql.so` module, the problem is probably that Perl cannot locate the `libmysqlclient.so` shared library. You should be able to fix this problem by one of the following methods:

- Copy `libmysqlclient.so` to the directory where your other shared libraries are located (probably `/usr/lib` or `/lib`).
- Modify the `-L` options used to compile `DBD::mysql` to reflect the actual location of `libmysqlclient.so`.
- On Linux, you can add the path name of the directory where `libmysqlclient.so` is located to the `/etc/ld.so.conf` file.
- Add the path name of the directory where `libmysqlclient.so` is located to the `LD_RUN_PATH` environment variable. Some systems use `LD_LIBRARY_PATH` instead.

Note that you may also need to modify the `-L` options if there are other libraries that the linker fails to find. For example, if the linker cannot find `libc` because it is in `/lib` and the link command specifies `-L/usr/lib`, change the `-L` option to `-L/lib` or add `-L/lib` to the existing link command.

If you get the following errors from `DBD::mysql`, you are probably using `gcc` (or using an old binary compiled with `gcc`):

```
/usr/bin/perl: can't resolve symbol '__moddi3'
/usr/bin/perl: can't resolve symbol '__divdi3'
```

Add `-L/usr/lib/gcc-lib/... -lgcc` to the link command when the `mysql.so` library gets built (check the output from `make` for `mysql.so` when you compile the Perl client). The `-L` option should specify the path name of the directory where `libgcc.a` is located on your system.

Another cause of this problem may be that Perl and MySQL are not both compiled with `gcc`. In this case, you can solve the mismatch by compiling both with `gcc`.

Chapter 3 Tutorial

Table of Contents

This chapter provides a tutorial introduction to MySQL by showing how to use the `mysql` client program to create and use a simple database. `mysql` (sometimes referred to as the “terminal monitor” or just “monitor”) is an interactive program that enables you to connect to a MySQL server, run queries, and view the results. `mysql` may also be used in batch mode: you place your queries in a file beforehand, then tell `mysql` to execute the contents of the file. Both ways of using `mysql` are covered here.

To see a list of options provided by `mysql`, invoke it with the `--help` option:

```
$> mysql --help
```

This chapter assumes that `mysql` is installed on your machine and that a MySQL server is available to which you can connect. If this is not true, contact your MySQL administrator. (If *you* are the administrator, you need to consult the relevant portions of this manual, such as Chapter 5, *MySQL Server Administration*.)

This chapter describes the entire process of setting up and using a database. If you are interested only in accessing an existing database, you may want to skip the sections that describe how to create the database and the tables it contains.

Because this chapter is tutorial in nature, many details are necessarily omitted. Consult the relevant sections of the manual for more information on the topics covered here.

3.1 Connecting to and Disconnecting from the Server

To connect to the server, you usually need to provide a MySQL user name when you invoke `mysql` and, most likely, a password. If the server runs on a machine other than the one where you log in, you must also specify a host name. Contact your administrator to find out what connection parameters you should use to connect (that is, what host, user name, and password to use). Once you know the proper parameters, you should be able to connect like this:

```
$> mysql -h host -u user -p
```

```
Enter password: ********
```

host and *user* represent the host name where your MySQL server is running and the user name of your MySQL account. Substitute appropriate values for your setup. The `********` represents your password; enter it when `mysql` displays the `Enter password:` prompt.

If that works, you should see some introductory information followed by a `mysql>` prompt:

```
$> mysql -h host -u user -p
Enter password: ********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 25338 to server version: 8.0.32-standard

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

The `mysql>` prompt tells you that `mysql` is ready for you to enter SQL statements.

If you are logging in on the same machine that MySQL is running on, you can omit the host, and simply use the following:

```
$> mysql -u user -p
```

If, when you attempt to log in, you get an error message such as `ERROR 2002 (HY000): Can't connect to local MySQL server through socket '/tmp/mysql.sock' (2)`, it means that the MySQL server daemon (Unix) or service (Windows) is not running. Consult the administrator or see the section of Chapter 2, *Installing and Upgrading MySQL* that is appropriate to your operating system.

For help with other problems often encountered when trying to log in, see Section B.3.2, “Common Errors When Using MySQL Programs”.

Some MySQL installations permit users to connect as the anonymous (unnamed) user to the server running on the local host. If this is the case on your machine, you should be able to connect to that server by invoking `mysql` without any options:

```
$> mysql
```

After you have connected successfully, you can disconnect any time by typing `QUIT` (or `\q`) at the `mysql>` prompt:

```
mysql> QUIT
Bye
```

On Unix, you can also disconnect by pressing Control+D.

Most examples in the following sections assume that you are connected to the server. They indicate this by the `mysql>` prompt.

3.2 Entering Queries

Make sure that you are connected to the server, as discussed in the previous section. Doing so does not in itself select any database to work with, but that is okay. At this point, it is more important to find out a little about how to issue queries than to jump right in creating tables, loading data into them, and retrieving data from them. This section describes the basic principles of entering queries, using several queries you can try out to familiarize yourself with how `mysql` works.

Here is a simple query that asks the server to tell you its version number and the current date. Type it in as shown here following the `mysql>` prompt and press Enter:

```
mysql> SELECT VERSION(), CURRENT_DATE;
+-----------+--------------+
| VERSION() | CURRENT_DATE |
```

```
+-----------+--------------+
| 5.8.0-m17 | 2015-12-21   |
+-----------+--------------+
1 row in set (0.02 sec)
mysql>
```

This query illustrates several things about `mysql`:

- A query normally consists of an SQL statement followed by a semicolon. (There are some exceptions where a semicolon may be omitted. `QUIT`, mentioned earlier, is one of them. We'll get to others later.)
- When you issue a query, `mysql` sends it to the server for execution and displays the results, then prints another `mysql>` prompt to indicate that it is ready for another query.
- `mysql` displays query output in tabular form (rows and columns). The first row contains labels for the columns. The rows following are the query results. Normally, column labels are the names of the columns you fetch from database tables. If you're retrieving the value of an expression rather than a table column (as in the example just shown), `mysql` labels the column using the expression itself.
- `mysql` shows how many rows were returned and how long the query took to execute, which gives you a rough idea of server performance. These values are imprecise because they represent wall clock time (not CPU or machine time), and because they are affected by factors such as server load and network latency. (For brevity, the “rows in set” line is sometimes not shown in the remaining examples in this chapter.)

Keywords may be entered in any lettercase. The following queries are equivalent:

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date;
mysql> SeLeCt vErSiOn(), current_DATE;
```

Here is another query. It demonstrates that you can use `mysql` as a simple calculator:

```
mysql> SELECT SIN(PI()/4), (4+1)*5;
+------------------+---------+
| SIN(PI()/4)      | (4+1)*5 |
+------------------+---------+
| 0.70710678118655 |      25 |
+------------------+---------+
1 row in set (0.02 sec)
```

The queries shown thus far have been relatively short, single-line statements. You can even enter multiple statements on a single line. Just end each one with a semicolon:

```
mysql> SELECT VERSION(); SELECT NOW();
+-----------+
| VERSION() |
+-----------+
| 8.0.13    |
+-----------+
1 row in set (0.00 sec)

+---------------------+
| NOW()               |
+---------------------+
| 2018-08-24 00:56:40 |
+---------------------+
1 row in set (0.00 sec)
```

A query need not be given all on a single line, so lengthy queries that require several lines are not a problem. `mysql` determines where your statement ends by looking for the terminating semicolon, not by looking for the end of the input line. (In other words, `mysql` accepts free-format input: it collects input lines but does not execute them until it sees the semicolon.)

Here is a simple multiple-line statement:

```
mysql> SELECT
    -> USER()
    -> ,
    -> CURRENT_DATE;
+---------------+--------------+
| USER()        | CURRENT_DATE |
+---------------+--------------+
| jon@localhost | 2018-08-24   |
+---------------+--------------+
```

In this example, notice how the prompt changes from `mysql>` to `->` after you enter the first line of a multiple-line query. This is how `mysql` indicates that it has not yet seen a complete statement and is waiting for the rest. The prompt is your friend, because it provides valuable feedback. If you use that feedback, you can always be aware of what `mysql` is waiting for.

If you decide you do not want to execute a query that you are in the process of entering, cancel it by typing `\c`:

```
mysql> SELECT
    -> USER()
    -> \c
mysql>
```

Here, too, notice the prompt. It switches back to `mysql>` after you type `\c`, providing feedback to indicate that `mysql` is ready for a new query.

The following table shows each of the prompts you may see and summarizes what they mean about the state that `mysql` is in.

Prompt	Meaning
`mysql>`	Ready for new query
`->`	Waiting for next line of multiple-line query
`'>`	Waiting for next line, waiting for completion of a string that began with a single quote (`'`)
`">`	Waiting for next line, waiting for completion of a string that began with a double quote (`"`)
`` `> ``	Waiting for next line, waiting for completion of an identifier that began with a backtick (`` ` ``)
`/*>`	Waiting for next line, waiting for completion of a comment that began with `/*`

Multiple-line statements commonly occur by accident when you intend to issue a query on a single line, but forget the terminating semicolon. In this case, `mysql` waits for more input:

```
mysql> SELECT USER()
    ->
```

If this happens to you (you think you've entered a statement but the only response is a `->` prompt), most likely `mysql` is waiting for the semicolon. If you don't notice what the prompt is telling you, you might sit there for a while before realizing what you need to do. Enter a semicolon to complete the statement, and `mysql` executes it:

```
mysql> SELECT USER()
    -> ;
+---------------+
| USER()        |
+---------------+
| jon@localhost |
+---------------+
```

The `'>` and `">` prompts occur during string collection (another way of saying that MySQL is waiting for completion of a string). In MySQL, you can write strings surrounded by either `'` or `"` characters (for

example, `'hello'` or `"goodbye"`), and `mysql` lets you enter strings that span multiple lines. When you see a `'>` or `">` prompt, it means that you have entered a line containing a string that begins with a `'` or `"` quote character, but have not yet entered the matching quote that terminates the string. This often indicates that you have inadvertently left out a quote character. For example:

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND age < 30;
    '>
```

If you enter this `SELECT` statement, then press **Enter** and wait for the result, nothing happens. Instead of wondering why this query takes so long, notice the clue provided by the `'>` prompt. It tells you that `mysql` expects to see the rest of an unterminated string. (Do you see the error in the statement? The string `'Smith` is missing the second single quotation mark.)

At this point, what do you do? The simplest thing is to cancel the query. However, you cannot just type `\c` in this case, because `mysql` interprets it as part of the string that it is collecting. Instead, enter the closing quote character (so `mysql` knows you've finished the string), then type `\c`:

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND age < 30;
    '> '\c
mysql>
```

The prompt changes back to `mysql>`, indicating that `mysql` is ready for a new query.

The `` `> `` prompt is similar to the `'>` and `">` prompts, but indicates that you have begun but not completed a backtick-quoted identifier.

It is important to know what the `'>`, `">`, and `` `> `` prompts signify, because if you mistakenly enter an unterminated string, any further lines you type appear to be ignored by `mysql`—including a line containing `QUIT`. This can be quite confusing, especially if you do not know that you need to supply the terminating quote before you can cancel the current query.

Note

Multiline statements from this point on are written without the secondary (`->` or other) prompts, to make it easier to copy and paste the statements to try for yourself.

3.3 Creating and Using a Database

Once you know how to enter SQL statements, you are ready to access a database.

Suppose that you have several pets in your home (your menagerie) and you would like to keep track of various types of information about them. You can do so by creating tables to hold your data and loading them with the desired information. Then you can answer different sorts of questions about your animals by retrieving data from the tables. This section shows you how to perform the following operations:

- Create a database
- Create a table
- Load data into the table
- Retrieve data from the table in various ways
- Use multiple tables

The menagerie database is simple (deliberately), but it is not difficult to think of real-world situations in which a similar type of database might be used. For example, a database like this could be used by a farmer to keep track of livestock, or by a veterinarian to keep track of patient records. A menagerie distribution containing some of the queries and sample data used in the following sections can be

obtained from the MySQL website. It is available in both compressed `tar` file and Zip formats at https://dev.mysql.com/doc/.

Use the `SHOW` statement to find out what databases currently exist on the server:

```
mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
| mysql    |
| test     |
| tmp      |
+----------+
```

The `mysql` database describes user access privileges. The `test` database often is available as a workspace for users to try things out.

The list of databases displayed by the statement may be different on your machine; `SHOW DATABASES` does not show databases that you have no privileges for if you do not have the `SHOW DATABASES` privilege. See Section 13.7.7.14, “SHOW DATABASES Statement”.

If the `test` database exists, try to access it:

```
mysql> USE test
Database changed
```

`USE`, like `QUIT`, does not require a semicolon. (You can terminate such statements with a semicolon if you like; it does no harm.) The `USE` statement is special in another way, too: it must be given on a single line.

You can use the `test` database (if you have access to it) for the examples that follow, but anything you create in that database can be removed by anyone else with access to it. For this reason, you should probably ask your MySQL administrator for permission to use a database of your own. Suppose that you want to call yours `menagerie`. The administrator needs to execute a statement like this:

```
mysql> GRANT ALL ON menagerie.* TO 'your_mysql_name'@'your_client_host';
```

where `your_mysql_name` is the MySQL user name assigned to you and `your_client_host` is the host from which you connect to the server.

3.3.1 Creating and Selecting a Database

If the administrator creates your database for you when setting up your permissions, you can begin using it. Otherwise, you need to create it yourself:

```
mysql> CREATE DATABASE menagerie;
```

Under Unix, database names are case-sensitive (unlike SQL keywords), so you must always refer to your database as `menagerie`, not as `Menagerie`, `MENAGERIE`, or some other variant. This is also true for table names. (Under Windows, this restriction does not apply, although you must refer to databases and tables using the same lettercase throughout a given query. However, for a variety of reasons, the recommended best practice is always to use the same lettercase that was used when the database was created.)

Note

If you get an error such as `ERROR 1044 (42000): Access denied for user 'micah'@'localhost' to database 'menagerie'` when attempting to create a database, this means that your user account does not have the necessary privileges to do so. Discuss this with the administrator or see Section 6.2, “Access Control and Account Management”.

Creating a database does not select it for use; you must do that explicitly. To make `menagerie` the current database, use this statement:

```
mysql> USE menagerie
Database changed
```

Your database needs to be created only once, but you must select it for use each time you begin a `mysql` session. You can do this by issuing a `USE` statement as shown in the example. Alternatively, you can select the database on the command line when you invoke `mysql`. Just specify its name after any connection parameters that you might need to provide. For example:

```
$> mysql -h host -u user -p menagerie
Enter password: ********
```

Important

`menagerie` in the command just shown is **not** your password. If you want to supply your password on the command line after the `-p` option, you must do so with no intervening space (for example, as `-ppassword`, not as `-p password`). However, putting your password on the command line is not recommended, because doing so exposes it to snooping by other users logged in on your machine.

Note

You can see at any time which database is currently selected using `SELECT DATABASE()`.

3.3.2 Creating a Table

Creating the database is the easy part, but at this point it is empty, as `SHOW TABLES` tells you:

```
mysql> SHOW TABLES;
Empty set (0.00 sec)
```

The harder part is deciding what the structure of your database should be: what tables you need and what columns should be in each of them.

You want a table that contains a record for each of your pets. This can be called the `pet` table, and it should contain, as a bare minimum, each animal's name. Because the name by itself is not very interesting, the table should contain other information. For example, if more than one person in your family keeps pets, you might want to list each animal's owner. You might also want to record some basic descriptive information such as species and sex.

How about age? That might be of interest, but it is not a good thing to store in a database. Age changes as time passes, which means you'd have to update your records often. Instead, it is better to store a fixed value such as date of birth. Then, whenever you need age, you can calculate it as the difference between the current date and the birth date. MySQL provides functions for doing date arithmetic, so this is not difficult. Storing birth date rather than age has other advantages, too:

- You can use the database for tasks such as generating reminders for upcoming pet birthdays. (If you think this type of query is somewhat silly, note that it is the same question you might ask in the context of a business database to identify clients to whom you need to send out birthday greetings in the current week or month, for that computer-assisted personal touch.)
- You can calculate age in relation to dates other than the current date. For example, if you store death date in the database, you can easily calculate how old a pet was when it died.

You can probably think of other types of information that would be useful in the `pet` table, but the ones identified so far are sufficient: name, owner, species, sex, birth, and death.

Use a `CREATE TABLE` statement to specify the layout of your table:

```
mysql> CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),
       species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);
```

`VARCHAR` is a good choice for the `name`, `owner`, and `species` columns because the column values vary in length. The lengths in those column definitions need not all be the same, and need not be `20`. You can normally pick any length from `1` to `65535`, whatever seems most reasonable to you. If you make a poor choice and it turns out later that you need a longer field, MySQL provides an `ALTER TABLE` statement.

Several types of values can be chosen to represent sex in animal records, such as `'m'` and `'f'`, or perhaps `'male'` and `'female'`. It is simplest to use the single characters `'m'` and `'f'`.

The use of the `DATE` data type for the `birth` and `death` columns is a fairly obvious choice.

Once you have created a table, `SHOW TABLES` should produce some output:

```
mysql> SHOW TABLES;
+--------------------+
| Tables in menagerie |
+--------------------+
| pet                |
+--------------------+
```

To verify that your table was created the way you expected, use a `DESCRIBE` statement:

```
mysql> DESCRIBE pet;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

You can use `DESCRIBE` any time, for example, if you forget the names of the columns in your table or what types they have.

For more information about MySQL data types, see Chapter 11, *Data Types*.

3.3.3 Loading Data into a Table

After creating your table, you need to populate it. The `LOAD DATA` and `INSERT` statements are useful for this.

Suppose that your pet records can be described as shown here. (Observe that MySQL expects dates in *`'YYYY-MM-DD'`* format; this may differ from what you are used to.)

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1993-02-04	
Claws	Gwen	cat	m	1994-03-17	
Buffy	Harold	dog	f	1989-05-13	
Fang	Benny	dog	m	1990-08-27	
Bowser	Diane	dog	m	1979-08-31	1995-07-29
Chirpy	Gwen	bird	f	1998-09-11	
Whistler	Gwen	bird		1997-12-09	
Slim	Benny	snake	m	1996-04-29	

Because you are beginning with an empty table, an easy way to populate it is to create a text file containing a row for each of your animals, then load the contents of the file into the table with a single statement.

You could create a text file `pet.txt` containing one record per line, with values separated by tabs, and given in the order in which the columns were listed in the `CREATE TABLE` statement. For missing values (such as unknown sexes or death dates for animals that are still living), you can use `NULL` values. To represent these in your text file, use `\N` (backslash, capital-N). For example, the record for Whistler the bird would look like this (where the whitespace between values is a single tab character):

```
Whistler        Gwen    bird    \N      1997-12-09      \N
```

To load the text file `pet.txt` into the `pet` table, use this statement:

```
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet;
```

If you created the file on Windows with an editor that uses `\r\n` as a line terminator, you should use this statement instead:

```
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet
       LINES TERMINATED BY '\r\n';
```

(On an Apple machine running macOS, you would likely want to use `LINES TERMINATED BY '\r'`.)

You can specify the column value separator and end of line marker explicitly in the `LOAD DATA` statement if you wish, but the defaults are tab and linefeed. These are sufficient for the statement to read the file `pet.txt` properly.

If the statement fails, it is likely that your MySQL installation does not have local file capability enabled by default. See Section 6.1.6, “Security Considerations for LOAD DATA LOCAL”, for information on how to change this.

When you want to add new records one at a time, the `INSERT` statement is useful. In its simplest form, you supply values for each column, in the order in which the columns were listed in the `CREATE TABLE` statement. Suppose that Diane gets a new hamster named “Puffball.” You could add a new record using an `INSERT` statement like this:

```
mysql> INSERT INTO pet
       VALUES ('Puffball','Diane','hamster','f','1999-03-30',NULL);
```

String and date values are specified as quoted strings here. Also, with `INSERT`, you can insert `NULL` directly to represent a missing value. You do not use `\N` like you do with `LOAD DATA`.

From this example, you should be able to see that there would be a lot more typing involved to load your records initially using several `INSERT` statements rather than a single `LOAD DATA` statement.

3.3.4 Retrieving Information from a Table

The `SELECT` statement is used to pull information from a table. The general form of the statement is:

```
SELECT what_to_select
FROM which_table
WHERE conditions_to_satisfy;
```

`what_to_select` indicates what you want to see. This can be a list of columns, or `*` to indicate “all columns.” *`which_table`* indicates the table from which you want to retrieve data. The `WHERE` clause is optional. If it is present, *`conditions_to_satisfy`* specifies one or more conditions that rows must satisfy to qualify for retrieval.

3.3.4.1 Selecting All Data

The simplest form of `SELECT` retrieves everything from a table:

```
mysql> SELECT * FROM pet;
+----------+--------+---------+------+------------+------------+
| name     | owner  | species | sex  | birth      | death      |
+----------+--------+---------+------+------------+------------+
```

```
| Fluffy   | Harold | cat     | f    | 1993-02-04 | NULL       |
| Claws    | Gwen   | cat     | m    | 1994-03-17 | NULL       |
| Buffy    | Harold | dog     | f    | 1989-05-13 | NULL       |
| Fang     | Benny  | dog     | m    | 1990-08-27 | NULL       |
| Bowser   | Diane  | dog     | m    | 1979-08-31 | 1995-07-29 |
| Chirpy   | Gwen   | bird    | f    | 1998-09-11 | NULL       |
| Whistler | Gwen   | bird    | NULL | 1997-12-09 | NULL       |
| Slim     | Benny  | snake   | m    | 1996-04-29 | NULL       |
| Puffball | Diane  | hamster | f    | 1999-03-30 | NULL       |
+----------+--------+---------+------+------------+------------+
```

This form of `SELECT` uses `*`, which is shorthand for “select all columns.” This is useful if you want to review your entire table, for example, after you've just loaded it with your initial data set. For example, you may happen to think that the birth date for Bowser doesn't seem quite right. Consulting your original pedigree papers, you find that the correct birth year should be 1989, not 1979.

There are at least two ways to fix this:

- Edit the file `pet.txt` to correct the error, then empty the table and reload it using `DELETE` and `LOAD DATA`:

  ```
  mysql> DELETE FROM pet;
  mysql> LOAD DATA LOCAL INFILE 'pet.txt' INTO TABLE pet;
  ```

  However, if you do this, you must also re-enter the record for Puffball.

- Fix only the erroneous record with an `UPDATE` statement:

  ```
  mysql> UPDATE pet SET birth = '1989-08-31' WHERE name = 'Bowser';
  ```

  The `UPDATE` changes only the record in question and does not require you to reload the table.

There is an exception to the principle that `SELECT *` selects all columns. If a table contains invisible columns, `*` does not include them. For more information, see Section 13.1.20.10, “Invisible Columns”.

3.3.4.2 Selecting Particular Rows

As shown in the preceding section, it is easy to retrieve an entire table. Just omit the `WHERE` clause from the `SELECT` statement. But typically you don't want to see the entire table, particularly when it becomes large. Instead, you're usually more interested in answering a particular question, in which case you specify some constraints on the information you want. Let's look at some selection queries in terms of questions about your pets that they answer.

You can select only particular rows from your table. For example, if you want to verify the change that you made to Bowser's birth date, select Bowser's record like this:

```
mysql> SELECT * FROM pet WHERE name = 'Bowser';
+--------+-------+---------+------+------------+------------+
| name   | owner | species | sex  | birth      | death      |
+--------+-------+---------+------+------------+------------+
| Bowser | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+-------+---------+------+------------+------------+
```

The output confirms that the year is correctly recorded as 1989, not 1979.

String comparisons normally are case-insensitive, so you can specify the name as `'bowser'`, `'BOWSER'`, and so forth. The query result is the same.

You can specify conditions on any column, not just `name`. For example, if you want to know which animals were born during or after 1998, test the `birth` column:

```
mysql> SELECT * FROM pet WHERE birth >= '1998-1-1';
+----------+-------+---------+------+------------+-------+
| name     | owner | species | sex  | birth      | death |
+----------+-------+---------+------+------------+-------+
```

```
| Chirpy   | Gwen  | bird    | f    | 1998-09-11 | NULL  |
| Puffball | Diane | hamster | f    | 1999-03-30 | NULL  |
+----------+-------+---------+------+------------+-------+
```

You can combine conditions, for example, to locate female dogs:

```
mysql> SELECT * FROM pet WHERE species = 'dog' AND sex = 'f';
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

The preceding query uses the `AND` logical operator. There is also an `OR` operator:

```
mysql> SELECT * FROM pet WHERE species = 'snake' OR species = 'bird';
+----------+-------+---------+------+------------+-------+
| name     | owner | species | sex  | birth      | death |
+----------+-------+---------+------+------------+-------+
| Chirpy   | Gwen  | bird    | f    | 1998-09-11 | NULL  |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL  |
| Slim     | Benny | snake   | m    | 1996-04-29 | NULL  |
+----------+-------+---------+------+------------+-------+
```

`AND` and `OR` may be intermixed, although `AND` has higher precedence than `OR`. If you use both operators, it is a good idea to use parentheses to indicate explicitly how conditions should be grouped:

```
mysql> SELECT * FROM pet WHERE (species = 'cat' AND sex = 'm')
       OR (species = 'dog' AND sex = 'f');
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

3.3.4.3 Selecting Particular Columns

If you do not want to see entire rows from your table, just name the columns in which you are interested, separated by commas. For example, if you want to know when your animals were born, select the `name` and `birth` columns:

```
mysql> SELECT name, birth FROM pet;
+----------+------------+
| name     | birth      |
+----------+------------+
| Fluffy   | 1993-02-04 |
| Claws    | 1994-03-17 |
| Buffy    | 1989-05-13 |
| Fang     | 1990-08-27 |
| Bowser   | 1989-08-31 |
| Chirpy   | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
| Puffball | 1999-03-30 |
+----------+------------+
```

To find out who owns pets, use this query:

```
mysql> SELECT owner FROM pet;
+--------+
| owner  |
+--------+
| Harold |
| Gwen   |
| Harold |
| Benny  |
| Diane  |
| Gwen   |
| Gwen   |
```

```
| Benny  |
| Diane  |
+--------+
```

Notice that the query simply retrieves the `owner` column from each record, and some of them appear more than once. To minimize the output, retrieve each unique output record just once by adding the keyword `DISTINCT`:

```
mysql> SELECT DISTINCT owner FROM pet;
+--------+
| owner  |
+--------+
| Benny  |
| Diane  |
| Gwen   |
| Harold |
+--------+
```

You can use a `WHERE` clause to combine row selection with column selection. For example, to get birth dates for dogs and cats only, use this query:

```
mysql> SELECT name, species, birth FROM pet
       WHERE species = 'dog' OR species = 'cat';
+--------+---------+------------+
| name   | species | birth      |
+--------+---------+------------+
| Fluffy | cat     | 1993-02-04 |
| Claws  | cat     | 1994-03-17 |
| Buffy  | dog     | 1989-05-13 |
| Fang   | dog     | 1990-08-27 |
| Bowser | dog     | 1989-08-31 |
+--------+---------+------------+
```

3.3.4.4 Sorting Rows

You may have noticed in the preceding examples that the result rows are displayed in no particular order. It is often easier to examine query output when the rows are sorted in some meaningful way. To sort a result, use an `ORDER BY` clause.

Here are animal birthdays, sorted by date:

```
mysql> SELECT name, birth FROM pet ORDER BY birth;
+----------+------------+
| name     | birth      |
+----------+------------+
| Buffy    | 1989-05-13 |
| Bowser   | 1989-08-31 |
| Fang     | 1990-08-27 |
| Fluffy   | 1993-02-04 |
| Claws    | 1994-03-17 |
| Slim     | 1996-04-29 |
| Whistler | 1997-12-09 |
| Chirpy   | 1998-09-11 |
| Puffball | 1999-03-30 |
+----------+------------+
```

On character type columns, sorting—like all other comparison operations—is normally performed in a case-insensitive fashion. This means that the order is undefined for columns that are identical except for their case. You can force a case-sensitive sort for a column by using `BINARY` like so: `ORDER BY BINARY col_name`.

The default sort order is ascending, with smallest values first. To sort in reverse (descending) order, add the `DESC` keyword to the name of the column you are sorting by:

```
mysql> SELECT name, birth FROM pet ORDER BY birth DESC;
+----------+------------+
| name     | birth      |
```

```
+----------+------------+
| Puffball | 1999-03-30 |
| Chirpy   | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
| Claws    | 1994-03-17 |
| Fluffy   | 1993-02-04 |
| Fang     | 1990-08-27 |
| Bowser   | 1989-08-31 |
| Buffy    | 1989-05-13 |
+----------+------------+
```

You can sort on multiple columns, and you can sort different columns in different directions. For example, to sort by type of animal in ascending order, then by birth date within animal type in descending order (youngest animals first), use the following query:

```
mysql> SELECT name, species, birth FROM pet
       ORDER BY species, birth DESC;
+----------+---------+------------+
| name     | species | birth      |
+----------+---------+------------+
| Chirpy   | bird    | 1998-09-11 |
| Whistler | bird    | 1997-12-09 |
| Claws    | cat     | 1994-03-17 |
| Fluffy   | cat     | 1993-02-04 |
| Fang     | dog     | 1990-08-27 |
| Bowser   | dog     | 1989-08-31 |
| Buffy    | dog     | 1989-05-13 |
| Puffball | hamster | 1999-03-30 |
| Slim     | snake   | 1996-04-29 |
+----------+---------+------------+
```

The `DESC` keyword applies only to the column name immediately preceding it (`birth`); it does not affect the `species` column sort order.

3.3.4.5 Date Calculations

MySQL provides several functions that you can use to perform calculations on dates, for example, to calculate ages or extract parts of dates.

To determine how many years old each of your pets is, use the `TIMESTAMPDIFF()` function. Its arguments are the unit in which you want the result expressed, and the two dates for which to take the difference. The following query shows, for each pet, the birth date, the current date, and the age in years. An *alias* (`age`) is used to make the final output column label more meaningful.

```
mysql> SELECT name, birth, CURDATE(),
       TIMESTAMPDIFF(YEAR,birth,CURDATE()) AS age
       FROM pet;
+----------+------------+------------+------+
| name     | birth      | CURDATE()  | age  |
+----------+------------+------------+------+
| Fluffy   | 1993-02-04 | 2003-08-19 |   10 |
| Claws    | 1994-03-17 | 2003-08-19 |    9 |
| Buffy    | 1989-05-13 | 2003-08-19 |   14 |
| Fang     | 1990-08-27 | 2003-08-19 |   12 |
| Bowser   | 1989-08-31 | 2003-08-19 |   13 |
| Chirpy   | 1998-09-11 | 2003-08-19 |    4 |
| Whistler | 1997-12-09 | 2003-08-19 |    5 |
| Slim     | 1996-04-29 | 2003-08-19 |    7 |
| Puffball | 1999-03-30 | 2003-08-19 |    4 |
+----------+------------+------------+------+
```

The query works, but the result could be scanned more easily if the rows were presented in some order. This can be done by adding an `ORDER BY name` clause to sort the output by name:

```
mysql> SELECT name, birth, CURDATE(),
       TIMESTAMPDIFF(YEAR,birth,CURDATE()) AS age
       FROM pet ORDER BY name;
```

```
+----------+------------+------------+------+
| name     | birth      | CURDATE()  | age  |
+----------+------------+------------+------+
| Bowser   | 1989-08-31 | 2003-08-19 |   13 |
| Buffy    | 1989-05-13 | 2003-08-19 |   14 |
| Chirpy   | 1998-09-11 | 2003-08-19 |    4 |
| Claws    | 1994-03-17 | 2003-08-19 |    9 |
| Fang     | 1990-08-27 | 2003-08-19 |   12 |
| Fluffy   | 1993-02-04 | 2003-08-19 |   10 |
| Puffball | 1999-03-30 | 2003-08-19 |    4 |
| Slim     | 1996-04-29 | 2003-08-19 |    7 |
| Whistler | 1997-12-09 | 2003-08-19 |    5 |
+----------+------------+------------+------+
```

To sort the output by `age` rather than `name`, just use a different `ORDER BY` clause:

```
mysql> SELECT name, birth, CURDATE(),
       TIMESTAMPDIFF(YEAR,birth,CURDATE()) AS age
       FROM pet ORDER BY age;
+----------+------------+------------+------+
| name     | birth      | CURDATE()  | age  |
+----------+------------+------------+------+
| Chirpy   | 1998-09-11 | 2003-08-19 |    4 |
| Puffball | 1999-03-30 | 2003-08-19 |    4 |
| Whistler | 1997-12-09 | 2003-08-19 |    5 |
| Slim     | 1996-04-29 | 2003-08-19 |    7 |
| Claws    | 1994-03-17 | 2003-08-19 |    9 |
| Fluffy   | 1993-02-04 | 2003-08-19 |   10 |
| Fang     | 1990-08-27 | 2003-08-19 |   12 |
| Bowser   | 1989-08-31 | 2003-08-19 |   13 |
| Buffy    | 1989-05-13 | 2003-08-19 |   14 |
+----------+------------+------------+------+
```

A similar query can be used to determine age at death for animals that have died. You determine which animals these are by checking whether the `death` value is `NULL`. Then, for those with non-`NULL` values, compute the difference between the `death` and `birth` values:

```
mysql> SELECT name, birth, death,
       TIMESTAMPDIFF(YEAR,birth,death) AS age
       FROM pet WHERE death IS NOT NULL ORDER BY age;
+--------+------------+------------+------+
| name   | birth      | death      | age  |
+--------+------------+------------+------+
| Bowser | 1989-08-31 | 1995-07-29 |    5 |
+--------+------------+------------+------+
```

The query uses `death IS NOT NULL` rather than `death <> NULL` because `NULL` is a special value that cannot be compared using the usual comparison operators. This is discussed later. See Section 3.3.4.6, “Working with NULL Values”.

What if you want to know which animals have birthdays next month? For this type of calculation, year and day are irrelevant; you simply want to extract the month part of the `birth` column. MySQL provides several functions for extracting parts of dates, such as `YEAR()`, `MONTH()`, and `DAYOFMONTH()`. `MONTH()` is the appropriate function here. To see how it works, run a simple query that displays the value of both `birth` and `MONTH(birth)`:

```
mysql> SELECT name, birth, MONTH(birth) FROM pet;
+----------+------------+--------------+
| name     | birth      | MONTH(birth) |
+----------+------------+--------------+
| Fluffy   | 1993-02-04 |            2 |
| Claws    | 1994-03-17 |            3 |
| Buffy    | 1989-05-13 |            5 |
| Fang     | 1990-08-27 |            8 |
| Bowser   | 1989-08-31 |            8 |
| Chirpy   | 1998-09-11 |            9 |
| Whistler | 1997-12-09 |           12 |
| Slim     | 1996-04-29 |            4 |
| Puffball | 1999-03-30 |            3 |
```

```
+----------+------------+-------------+
```

Finding animals with birthdays in the upcoming month is also simple. Suppose that the current month is April. Then the month value is `4` and you can look for animals born in May (month `5`) like this:

```
mysql> SELECT name, birth FROM pet WHERE MONTH(birth) = 5;
+-------+------------+
| name  | birth      |
+-------+------------+
| Buffy | 1989-05-13 |
+-------+------------+
```

There is a small complication if the current month is December. You cannot merely add one to the month number (`12`) and look for animals born in month `13`, because there is no such month. Instead, you look for animals born in January (month `1`).

You can write the query so that it works no matter what the current month is, so that you do not have to use the number for a particular month. `DATE_ADD()` enables you to add a time interval to a given date. If you add a month to the value of `CURDATE()`, then extract the month part with `MONTH()`, the result produces the month in which to look for birthdays:

```
mysql> SELECT name, birth FROM pet
       WHERE MONTH(birth) = MONTH(DATE_ADD(CURDATE(),INTERVAL 1 MONTH));
```

A different way to accomplish the same task is to add `1` to get the next month after the current one after using the modulo function (`MOD`) to wrap the month value to `0` if it is currently `12`:

```
mysql> SELECT name, birth FROM pet
       WHERE MONTH(birth) = MOD(MONTH(CURDATE()), 12) + 1;
```

`MONTH()` returns a number between `1` and `12`. And `MOD(something,12)` returns a number between `0` and `11`. So the addition has to be after the `MOD()`, otherwise we would go from November (`11`) to January (`1`).

If a calculation uses invalid dates, the calculation fails and produces warnings:

```
mysql> SELECT '2018-10-31' + INTERVAL 1 DAY;
+-----------------------------+
| '2018-10-31' + INTERVAL 1 DAY |
+-----------------------------+
| 2018-11-01                  |
+-----------------------------+
mysql> SELECT '2018-10-32' + INTERVAL 1 DAY;
+-----------------------------+
| '2018-10-32' + INTERVAL 1 DAY |
+-----------------------------+
| NULL                        |
+-----------------------------+
mysql> SHOW WARNINGS;
+---------+------+----------------------------------------+
| Level   | Code | Message                                |
+---------+------+----------------------------------------+
| Warning | 1292 | Incorrect datetime value: '2018-10-32' |
+---------+------+----------------------------------------+
```

3.3.4.6 Working with NULL Values

The `NULL` value can be surprising until you get used to it. Conceptually, `NULL` means “a missing unknown value” and it is treated somewhat differently from other values.

To test for `NULL`, use the `IS NULL` and `IS NOT NULL` operators, as shown here:

```
mysql> SELECT 1 IS NULL, 1 IS NOT NULL;
+-----------+---------------+
| 1 IS NULL | 1 IS NOT NULL |
+-----------+---------------+
|         0 |             1 |
```

```
+-----------+--------------+
```

You cannot use arithmetic comparison operators such as `=`, `<`, or `<>` to test for `NULL`. To demonstrate this for yourself, try the following query:

```
mysql> SELECT 1 = NULL, 1 <> NULL, 1 < NULL, 1 > NULL;
+----------+-----------+----------+----------+
| 1 = NULL | 1 <> NULL | 1 < NULL | 1 > NULL |
+----------+-----------+----------+----------+
|     NULL |      NULL |     NULL |     NULL |
+----------+-----------+----------+----------+
```

Because the result of any arithmetic comparison with `NULL` is also `NULL`, you cannot obtain any meaningful results from such comparisons.

In MySQL, `0` or `NULL` means false and anything else means true. The default truth value from a boolean operation is `1`.

This special treatment of `NULL` is why, in the previous section, it was necessary to determine which animals are no longer alive using `death IS NOT NULL` instead of `death <> NULL`.

Two `NULL` values are regarded as equal in a `GROUP BY`.

When doing an `ORDER BY`, `NULL` values are presented first if you do `ORDER BY ... ASC` and last if you do `ORDER BY ... DESC`.

A common error when working with `NULL` is to assume that it is not possible to insert a zero or an empty string into a column defined as `NOT NULL`, but this is not the case. These are in fact values, whereas `NULL` means “not having a value.” You can test this easily enough by using `IS [NOT] NULL` as shown:

```
mysql> SELECT 0 IS NULL, 0 IS NOT NULL, '' IS NULL, '' IS NOT NULL;
+-----------+---------------+------------+----------------+
| 0 IS NULL | 0 IS NOT NULL | '' IS NULL | '' IS NOT NULL |
+-----------+---------------+------------+----------------+
|         0 |             1 |          0 |              1 |
+-----------+---------------+------------+----------------+
```

Thus it is entirely possible to insert a zero or empty string into a `NOT NULL` column, as these are in fact `NOT NULL`. See Section B.3.4.3, “Problems with NULL Values”.

3.3.4.7 Pattern Matching

MySQL provides standard SQL pattern matching as well as a form of pattern matching based on extended regular expressions similar to those used by Unix utilities such as `vi`, `grep`, and `sed`.

SQL pattern matching enables you to use `_` to match any single character and `%` to match an arbitrary number of characters (including zero characters). In MySQL, SQL patterns are case-insensitive by default. Some examples are shown here. Do not use `=` or `<>` when you use SQL patterns. Use the `LIKE` or `NOT LIKE` comparison operators instead.

To find names beginning with `b`:

```
mysql> SELECT * FROM pet WHERE name LIKE 'b%';
+--------+--------+---------+------+------------+------------+
| name   | owner  | species | sex  | birth      | death      |
+--------+--------+---------+------+------------+------------+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+--------+---------+------+------------+------------+
```

To find names ending with `fy`:

```
mysql> SELECT * FROM pet WHERE name LIKE '%fy';
+--------+--------+---------+------+------------+-------+
```

```
| name   | owner  | species | sex  | birth      | death |
+--------+--------+---------+------+------------+-------+
| Fluffy | Harold | cat     | f    | 1993-02-04 | NULL  |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL  |
+--------+--------+---------+------+------------+-------+
```

To find names containing a `w`:

```
mysql> SELECT * FROM pet WHERE name LIKE '%w%';
+----------+-------+---------+------+------------+------------+
| name     | owner | species | sex  | birth      | death      |
+----------+-------+---------+------+------------+------------+
| Claws    | Gwen  | cat     | m    | 1994-03-17 | NULL       |
| Bowser   | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL       |
+----------+-------+---------+------+------------+------------+
```

To find names containing exactly five characters, use five instances of the `_` pattern character:

```
mysql> SELECT * FROM pet WHERE name LIKE '_____';
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

The other type of pattern matching provided by MySQL uses extended regular expressions. When you test for a match for this type of pattern, use the `REGEXP_LIKE()` function (or the `REGEXP` or `RLIKE` operators, which are synonyms for `REGEXP_LIKE()`).

The following list describes some characteristics of extended regular expressions:

- `.` matches any single character.
- A character class `[...]` matches any character within the brackets. For example, `[abc]` matches `a`, `b`, or `c`. To name a range of characters, use a dash. `[a-z]` matches any letter, whereas `[0-9]` matches any digit.
- `*` matches zero or more instances of the thing preceding it. For example, `x*` matches any number of `x` characters, `[0-9]*` matches any number of digits, and `.*` matches any number of anything.
- A regular expression pattern match succeeds if the pattern matches anywhere in the value being tested. (This differs from a `LIKE` pattern match, which succeeds only if the pattern matches the entire value.)
- To anchor a pattern so that it must match the beginning or end of the value being tested, use `^` at the beginning or `$` at the end of the pattern.

To demonstrate how extended regular expressions work, the `LIKE` queries shown previously are rewritten here to use `REGEXP_LIKE()`.

To find names beginning with `b`, use `^` to match the beginning of the name:

```
mysql> SELECT * FROM pet WHERE REGEXP_LIKE(name, '^b');
+--------+--------+---------+------+------------+------------+
| name   | owner  | species | sex  | birth      | death      |
+--------+--------+---------+------+------------+------------+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1979-08-31 | 1995-07-29 |
+--------+--------+---------+------+------------+------------+
```

To force a regular expression comparison to be case-sensitive, use a case-sensitive collation, or use the `BINARY` keyword to make one of the strings a binary string, or specify the `c` match-control character. Each of these queries matches only lowercase `b` at the beginning of a name:

```
SELECT * FROM pet WHERE REGEXP_LIKE(name, '^b' COLLATE utf8mb4_0900_as_cs);
```

```
SELECT * FROM pet WHERE REGEXP_LIKE(name, BINARY '^b');
SELECT * FROM pet WHERE REGEXP_LIKE(name, '^b', 'c');
```

To find names ending with `fy`, use `$` to match the end of the name:

```
mysql> SELECT * FROM pet WHERE REGEXP_LIKE(name, 'fy$');
+--------+--------+---------+------+------------+-------+
| name   | owner  | species | sex  | birth      | death |
+--------+--------+---------+------+------------+-------+
| Fluffy | Harold | cat     | f    | 1993-02-04 | NULL  |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL  |
+--------+--------+---------+------+------------+-------+
```

To find names containing a `w`, use this query:

```
mysql> SELECT * FROM pet WHERE REGEXP_LIKE(name, 'w');
+----------+-------+---------+------+------------+------------+
| name     | owner | species | sex  | birth      | death      |
+----------+-------+---------+------+------------+------------+
| Claws    | Gwen  | cat     | m    | 1994-03-17 | NULL       |
| Bowser   | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL       |
+----------+-------+---------+------+------------+------------+
```

Because a regular expression pattern matches if it occurs anywhere in the value, it is not necessary in the previous query to put a wildcard on either side of the pattern to get it to match the entire value as would be true with an SQL pattern.

To find names containing exactly five characters, use `^` and `$` to match the beginning and end of the name, and five instances of `.` in between:

```
mysql> SELECT * FROM pet WHERE REGEXP_LIKE(name, '^.....$');
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

You could also write the previous query using the `{`*n*`}` (“repeat-*n*-times”) operator:

```
mysql> SELECT * FROM pet WHERE REGEXP_LIKE(name, '^.{5}$');
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

For more information about the syntax for regular expressions, see Section 12.8.2, “Regular Expressions”.

3.3.4.8 Counting Rows

Databases are often used to answer the question, “How often does a certain type of data occur in a table?” For example, you might want to know how many pets you have, or how many pets each owner has, or you might want to perform various kinds of census operations on your animals.

Counting the total number of animals you have is the same question as “How many rows are in the `pet` table?” because there is one record per pet. `COUNT(*)` counts the number of rows, so the query to count your animals looks like this:

```
mysql> SELECT COUNT(*) FROM pet;
+----------+
| COUNT(*) |
+----------+
|        9 |
```

```
+----------+
```

Earlier, you retrieved the names of the people who owned pets. You can use `COUNT()` if you want to find out how many pets each owner has:

```
mysql> SELECT owner, COUNT(*) FROM pet GROUP BY owner;
+--------+----------+
| owner  | COUNT(*) |
+--------+----------+
| Benny  |        2 |
| Diane  |        2 |
| Gwen   |        3 |
| Harold |        2 |
+--------+----------+
```

The preceding query uses `GROUP BY` to group all records for each `owner`. The use of `COUNT()` in conjunction with `GROUP BY` is useful for characterizing your data under various groupings. The following examples show different ways to perform animal census operations.

Number of animals per species:

```
mysql> SELECT species, COUNT(*) FROM pet GROUP BY species;
+---------+----------+
| species | COUNT(*) |
+---------+----------+
| bird    |        2 |
| cat     |        2 |
| dog     |        3 |
| hamster |        1 |
| snake   |        1 |
+---------+----------+
```

Number of animals per sex:

```
mysql> SELECT sex, COUNT(*) FROM pet GROUP BY sex;
+------+----------+
| sex  | COUNT(*) |
+------+----------+
| NULL |        1 |
| f    |        4 |
| m    |        4 |
+------+----------+
```

(In this output, `NULL` indicates that the sex is unknown.)

Number of animals per combination of species and sex:

```
mysql> SELECT species, sex, COUNT(*) FROM pet GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
+---------+------+----------+
| bird    | NULL |        1 |
| bird    | f    |        1 |
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
| dog     | m    |        2 |
| hamster | f    |        1 |
| snake   | m    |        1 |
+---------+------+----------+
```

You need not retrieve an entire table when you use `COUNT()`. For example, the previous query, when performed just on dogs and cats, looks like this:

```
mysql> SELECT species, sex, COUNT(*) FROM pet
       WHERE species = 'dog' OR species = 'cat'
       GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
```

```
+---------+------+----------+
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
| dog     | m    |        2 |
+---------+------+----------+
```

Or, if you wanted the number of animals per sex only for animals whose sex is known:

```
mysql> SELECT species, sex, COUNT(*) FROM pet
       WHERE sex IS NOT NULL
       GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
+---------+------+----------+
| bird    | f    |        1 |
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
| dog     | m    |        2 |
| hamster | f    |        1 |
| snake   | m    |        1 |
+---------+------+----------+
```

If you name columns to select in addition to the `COUNT()` value, a `GROUP BY` clause should be present that names those same columns. Otherwise, the following occurs:

- If the `ONLY_FULL_GROUP_BY` SQL mode is enabled, an error occurs:

```
mysql> SET sql_mode = 'ONLY_FULL_GROUP_BY';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT owner, COUNT(*) FROM pet;
ERROR 1140 (42000): In aggregated query without GROUP BY, expression
#1 of SELECT list contains nonaggregated column 'menagerie.pet.owner';
this is incompatible with sql_mode=only_full_group_by
```

- If `ONLY_FULL_GROUP_BY` is not enabled, the query is processed by treating all rows as a single group, but the value selected for each named column is nondeterministic. The server is free to select the value from any row:

```
mysql> SET sql_mode = '';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT owner, COUNT(*) FROM pet;
+--------+----------+
| owner  | COUNT(*) |
+--------+----------+
| Harold |        8 |
+--------+----------+
1 row in set (0.00 sec)
```

See also Section 12.20.3, “MySQL Handling of GROUP BY”. See Section 12.20.1, “Aggregate Function Descriptions” for information about `COUNT(expr)` behavior and related optimizations.

3.3.4.9 Using More Than one Table

The `pet` table keeps track of which pets you have. If you want to record other information about them, such as events in their lives like visits to the vet or when litters are born, you need another table. What should this table look like? It needs to contain the following information:

- The pet name so that you know which animal each event pertains to.
- A date so that you know when the event occurred.
- A field to describe the event.
- An event type field, if you want to be able to categorize events.

Given these considerations, the `CREATE TABLE` statement for the `event` table might look like this:

```
mysql> CREATE TABLE event (name VARCHAR(20), date DATE,
       type VARCHAR(15), remark VARCHAR(255));
```

As with the `pet` table, it is easiest to load the initial records by creating a tab-delimited text file containing the following information.

name	date	type	remark
Fluffy	1995-05-15	litter	4 kittens, 3 female, 1 male
Buffy	1993-06-23	litter	5 puppies, 2 female, 3 male
Buffy	1994-06-19	litter	3 puppies, 3 female
Chirpy	1999-03-21	vet	needed beak straightened
Slim	1997-08-03	vet	broken rib
Bowser	1991-10-12	kennel	
Fang	1991-10-12	kennel	
Fang	1998-08-28	birthday	Gave him a new chew toy
Claws	1998-03-17	birthday	Gave him a new flea collar
Whistler	1998-12-09	birthday	First birthday

Load the records like this:

```
mysql> LOAD DATA LOCAL INFILE 'event.txt' INTO TABLE event;
```

Based on what you have learned from the queries that you have run on the `pet` table, you should be able to perform retrievals on the records in the `event` table; the principles are the same. But when is the `event` table by itself insufficient to answer questions you might ask?

Suppose that you want to find out the ages at which each pet had its litters. We saw earlier how to calculate ages from two dates. The litter date of the mother is in the `event` table, but to calculate her age on that date you need her birth date, which is stored in the `pet` table. This means the query requires both tables:

```
mysql> SELECT pet.name,
       TIMESTAMPDIFF(YEAR,birth,date) AS age,
       remark
       FROM pet INNER JOIN event
         ON pet.name = event.name
       WHERE event.type = 'litter';
+--------+------+-----------------------------+
| name   | age  | remark                      |
+--------+------+-----------------------------+
| Fluffy |    2 | 4 kittens, 3 female, 1 male |
| Buffy  |    4 | 5 puppies, 2 female, 3 male |
| Buffy  |    5 | 3 puppies, 3 female         |
+--------+------+-----------------------------+
```

There are several things to note about this query:

- The `FROM` clause joins two tables because the query needs to pull information from both of them.
- When combining (joining) information from multiple tables, you need to specify how records in one table can be matched to records in the other. This is easy because they both have a `name` column. The query uses an `ON` clause to match up records in the two tables based on the `name` values.

The query uses an `INNER JOIN` to combine the tables. An `INNER JOIN` permits rows from either table to appear in the result if and only if both tables meet the conditions specified in the `ON` clause. In this example, the `ON` clause specifies that the `name` column in the `pet` table must match the `name` column in the `event` table. If a name appears in one table but not the other, the row does not appear in the result because the condition in the `ON` clause fails.

- Because the `name` column occurs in both tables, you must be specific about which table you mean when referring to the column. This is done by prepending the table name to the column name.

You need not have two different tables to perform a join. Sometimes it is useful to join a table to itself, if you want to compare records in a table to other records in that same table. For example, to find breeding pairs among your pets, you can join the `pet` table with itself to produce candidate pairs of live males and females of like species:

```
mysql> SELECT p1.name, p1.sex, p2.name, p2.sex, p1.species
       FROM pet AS p1 INNER JOIN pet AS p2
         ON p1.species = p2.species
         AND p1.sex = 'f' AND p1.death IS NULL
         AND p2.sex = 'm' AND p2.death IS NULL;
+--------+------+-------+------+---------+
| name   | sex  | name  | sex  | species |
+--------+------+-------+------+---------+
| Fluffy | f    | Claws | m    | cat     |
| Buffy  | f    | Fang  | m    | dog     |
+--------+------+-------+------+---------+
```

In this query, we specify aliases for the table name to refer to the columns and keep straight which instance of the table each column reference is associated with.

3.4 Getting Information About Databases and Tables

What if you forget the name of a database or table, or what the structure of a given table is (for example, what its columns are called)? MySQL addresses this problem through several statements that provide information about the databases and tables it supports.

You have previously seen `SHOW DATABASES`, which lists the databases managed by the server. To find out which database is currently selected, use the `DATABASE()` function:

```
mysql> SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
| menagerie  |
+------------+
```

If you have not yet selected any database, the result is `NULL`.

To find out what tables the default database contains (for example, when you are not sure about the name of a table), use this statement:

```
mysql> SHOW TABLES;
+----------------------+
| Tables_in_menagerie |
+----------------------+
| event                |
| pet                  |
+----------------------+
```

The name of the column in the output produced by this statement is always `Tables_in_`*`db_name`*, where *`db_name`* is the name of the database. See Section 13.7.7.39, “SHOW TABLES Statement”, for more information.

If you want to find out about the structure of a table, the `DESCRIBE` statement is useful; it displays information about each of a table's columns:

```
mysql> DESCRIBE pet;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

`Field` indicates the column name, `Type` is the data type for the column, `NULL` indicates whether the column can contain `NULL` values, `Key` indicates whether the column is indexed, and `Default` specifies the column's default value. `Extra` displays special information about columns: If a column was created with the `AUTO_INCREMENT` option, the value is `auto_increment` rather than empty.

`DESC` is a short form of `DESCRIBE`. See Section 13.8.1, “DESCRIBE Statement”, for more information.

You can obtain the `CREATE TABLE` statement necessary to create an existing table using the `SHOW CREATE TABLE` statement. See Section 13.7.7.10, “SHOW CREATE TABLE Statement”.

If you have indexes on a table, `SHOW INDEX FROM tbl_name` produces information about them. See Section 13.7.7.22, “SHOW INDEX Statement”, for more about this statement.

3.5 Using mysql in Batch Mode

In the previous sections, you used `mysql` interactively to enter statements and view the results. You can also run `mysql` in batch mode. To do this, put the statements you want to run in a file, then tell `mysql` to read its input from the file:

```
$> mysql < batch-file
```

If you are running `mysql` under Windows and have some special characters in the file that cause problems, you can do this:

```
C:\> mysql -e "source batch-file"
```

If you need to specify connection parameters on the command line, the command might look like this:

```
$> mysql -h host -u user -p < batch-file
Enter password: ********
```

When you use `mysql` this way, you are creating a script file, then executing the script.

If you want the script to continue even if some of the statements in it produce errors, you should use the `--force` command-line option.

Why use a script? Here are a few reasons:

- If you run a query repeatedly (say, every day or every week), making it a script enables you to avoid retyping it each time you execute it.
- You can generate new queries from existing ones that are similar by copying and editing script files.
- Batch mode can also be useful while you're developing a query, particularly for multiple-line statements or multiple-statement sequences. If you make a mistake, you don't have to retype everything. Just edit your script to correct the error, then tell `mysql` to execute it again.
- If you have a query that produces a lot of output, you can run the output through a pager rather than watching it scroll off the top of your screen:

  ```
  $> mysql < batch-file | more
  ```

- You can catch the output in a file for further processing:

```
$> mysql < batch-file > mysql.out
```

- You can distribute your script to other people so that they can also run the statements.
- Some situations do not allow for interactive use, for example, when you run a query from a `cron` job. In this case, you must use batch mode.

The default output format is different (more concise) when you run `mysql` in batch mode than when you use it interactively. For example, the output of `SELECT DISTINCT species FROM pet` looks like this when `mysql` is run interactively:

```
+---------+
| species |
+---------+
| bird    |
| cat     |
| dog     |
| hamster |
| snake   |
+---------+
```

In batch mode, the output looks like this instead:

```
species
bird
cat
dog
hamster
snake
```

If you want to get the interactive output format in batch mode, use `mysql -t`. To echo to the output the statements that are executed, use `mysql -v`.

You can also use scripts from the `mysql` prompt by using the `source` command or `\.` command:

```
mysql> source filename;
mysql> \. filename
```

See Section 4.5.1.5, “Executing SQL Statements from a Text File”, for more information.

3.6 Examples of Common Queries

Here are examples of how to solve some common problems with MySQL.

Some of the examples use the table `shop` to hold the price of each article (item number) for certain traders (dealers). Supposing that each trader has a single fixed price per article, then (`article`, `dealer`) is a primary key for the records.

Start the command-line tool `mysql` and select a database:

```
$> mysql your-database-name
```

To create and populate the example table, use these statements:

```
CREATE TABLE shop (
    article INT UNSIGNED  DEFAULT '0000' NOT NULL,
    dealer  CHAR(20)      DEFAULT ''     NOT NULL,
    price   DECIMAL(16,2) DEFAULT '0.00' NOT NULL,
    PRIMARY KEY(article, dealer));
INSERT INTO shop VALUES
    (1,'A',3.45),(1,'B',3.99),(2,'A',10.99),(3,'B',1.45),
    (3,'C',1.69),(3,'D',1.25),(4,'D',19.95);
```

After issuing the statements, the table should have the following contents:

```
SELECT * FROM shop ORDER BY article;

+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|       1 | A      |  3.45 |
|       1 | B      |  3.99 |
|       2 | A      | 10.99 |
|       3 | B      |  1.45 |
|       3 | C      |  1.69 |
|       3 | D      |  1.25 |
|       4 | D      | 19.95 |
+---------+--------+-------+
```

3.6.1 The Maximum Value for a Column

"What is the highest item number?"

```
SELECT MAX(article) AS article FROM shop;

+---------+
| article |
+---------+
|       4 |
+---------+
```

3.6.2 The Row Holding the Maximum of a Certain Column

Task: Find the number, dealer, and price of the most expensive article.

This is easily done with a subquery:

```
SELECT article, dealer, price
FROM   shop
WHERE  price=(SELECT MAX(price) FROM shop);

+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

Another solution is to use a `LEFT JOIN`, as shown here:

```
SELECT s1.article, s1.dealer, s1.price
FROM shop s1
LEFT JOIN shop s2 ON s1.price < s2.price
WHERE s2.article IS NULL;
```

You can also do this by sorting all rows descending by price and get only the first row using the MySQL-specific `LIMIT` clause, like this:

```
SELECT article, dealer, price
FROM shop
ORDER BY price DESC
LIMIT 1;
```

Note

If there were several most expensive articles, each with a price of 19.95, the `LIMIT` solution would show only one of them.

3.6.3 Maximum of Column per Group

Task: Find the highest price per article.

```
SELECT article, MAX(price) AS price
```

```
FROM   shop
GROUP BY article
ORDER BY article;

+---------+-------+
| article | price |
+---------+-------+
|    0001 |  3.99 |
|    0002 | 10.99 |
|    0003 |  1.69 |
|    0004 | 19.95 |
+---------+-------+
```

3.6.4 The Rows Holding the Group-wise Maximum of a Certain Column

Task: For each article, find the dealer or dealers with the most expensive price.

This problem can be solved with a subquery like this one:

```
SELECT article, dealer, price
FROM   shop s1
WHERE  price=(SELECT MAX(s2.price)
              FROM shop s2
              WHERE s1.article = s2.article)
ORDER BY article;

+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0001 | B      |  3.99 |
|    0002 | A      | 10.99 |
|    0003 | C      |  1.69 |
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

The preceding example uses a correlated subquery, which can be inefficient (see Section 13.2.15.7, “Correlated Subqueries”). Other possibilities for solving the problem are to use an uncorrelated subquery in the `FROM` clause, a `LEFT JOIN`, or a common table expression with a window function.

Uncorrelated subquery:

```
SELECT s1.article, dealer, s1.price
FROM shop s1
JOIN (
  SELECT article, MAX(price) AS price
  FROM shop
  GROUP BY article) AS s2
  ON s1.article = s2.article AND s1.price = s2.price
ORDER BY article;
```

`LEFT JOIN`:

```
SELECT s1.article, s1.dealer, s1.price
FROM shop s1
LEFT JOIN shop s2 ON s1.article = s2.article AND s1.price < s2.price
WHERE s2.article IS NULL
ORDER BY s1.article;
```

The `LEFT JOIN` works on the basis that when `s1.price` is at its maximum value, there is no `s2.price` with a greater value and thus the corresponding `s2.article` value is `NULL`. See Section 13.2.13.2, “JOIN Clause”.

Common table expression with window function:

```
WITH s1 AS (
   SELECT article, dealer, price,
          RANK() OVER (PARTITION BY article
                       ORDER BY price DESC
                      ) AS `Rank`
```

```
    FROM shop
)
SELECT article, dealer, price
  FROM s1
  WHERE `Rank` = 1
ORDER BY article;
```

3.6.5 Using User-Defined Variables

You can employ MySQL user variables to remember results without having to store them in temporary variables in the client. (See Section 9.4, “User-Defined Variables”.)

For example, to find the articles with the highest and lowest price you can do this:

```
mysql> SELECT @min_price:=MIN(price),@max_price:=MAX(price) FROM shop;
mysql> SELECT * FROM shop WHERE price=@min_price OR price=@max_price;
+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0003 | D      |  1.25 |
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

Note

It is also possible to store the name of a database object such as a table or a column in a user variable and then to use this variable in an SQL statement; however, this requires the use of a prepared statement. See Section 13.5, “Prepared Statements”, for more information.

3.6.6 Using Foreign Keys

MySQL supports foreign keys, which permit cross-referencing related data across tables, and foreign key constraints, which help keep the related data consistent.

A foreign key relationship involves a parent table that holds the initial column values, and a child table with column values that reference the parent column values. A foreign key constraint is defined on the child table.

This following example relates `parent` and `child` tables through a single-column foreign key and shows how a foreign key constraint enforces referential integrity.

Create the parent and child tables:

```
CREATE TABLE parent (
    id INT NOT NULL,
    PRIMARY KEY (id)
) ENGINE=INNODB;


CREATE TABLE child (
    id INT,
    parent_id INT,
    INDEX par_ind (parent_id),
    FOREIGN KEY (parent_id)
        REFERENCES parent(id)
) ENGINE=INNODB;
```

Insert a row into the parent table:

```
mysql> INSERT INTO parent (id) VALUES (1);
```

Verify that the data was inserted:

```
mysql> SELECT * FROM parent;
+----+
```

```
| id |
+----+
|  1 |
+----+
```

Insert a row into the child table:

```
mysql> INSERT INTO child (id,parent_id) VALUES (1,1);
```

The insert operation is successful because `parent_id` 1 is present in the parent table.

Insert a row into the child table with a `parent_id` value that is not present in the parent table:

```
mysql> INSERT INTO child (id,parent_id) VALUES(2,2);
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails
(`test`.`child`, CONSTRAINT `child_ibfk_1` FOREIGN KEY (`parent_id`)
REFERENCES `parent` (`id`))
```

The operation fails because the specified `parent_id` value does not exist in the parent table.

Try to delete the previously inserted row from the parent table:

```
mysql> DELETE FROM parent WHERE id VALUES = 1;
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails
(`test`.`child`, CONSTRAINT `child_ibfk_1` FOREIGN KEY (`parent_id`)
REFERENCES `parent` (`id`))
```

This operation fails because the record in the child table contains the referenced id (`parent_id`) value.

When an operation affects a key value in the parent table that has matching rows in the child table, the result depends on the referential action specified by `ON UPDATE` and `ON DELETE` subclauses of the `FOREIGN KEY` clause. Omitting `ON DELETE` and `ON UPDATE` clauses (as in the current child table definition) is the same as specifying the `RESTRICT` option, which rejects operations that affect a key value in the parent table that has matching rows in the parent table.

To demonstrate `ON DELETE` and `ON UPDATE` referential actions, drop the child table and recreate it to include `ON UPDATE` and `ON DELETE` subclauses with the `CASCADE` option. The `CASCADE` option automatically deletes or updates matching rows in the child table when deleting or updating rows in the parent table.

```
DROP TABLE child;

CREATE TABLE child (
    id INT,
    parent_id INT,
    INDEX par_ind (parent_id),
    FOREIGN KEY (parent_id)
        REFERENCES parent(id)
        ON UPDATE CASCADE
        ON DELETE CASCADE
) ENGINE=INNODB;
```

Insert the following rows into the child table:

```
mysql> INSERT INTO child (id,parent_id) VALUES(1,1),(2,1),(3,1);
```

Verify that the data was inserted:

```
mysql> SELECT * FROM child;
+------+-----------+
| id   | parent_id |
+------+-----------+
|    1 |         1 |
|    2 |         1 |
|    3 |         1 |
+------+-----------+
```

Update the id in the parent table, changing it from 1 to 2.

```
mysql> UPDATE parent SET id = 2 WHERE id = 1;
```

Verify that the update was successful:

```
mysql> SELECT * FROM parent;
+----+
| id |
+----+
|  2 |
+----+
```

Verify that the `ON UPDATE CASCADE` referential action updated the child table:

```
mysql> SELECT * FROM child;
+------+-----------+
| id   | parent_id |
+------+-----------+
|    1 |         2 |
|    2 |         2 |
|    3 |         2 |
+------+-----------+
```

To demonstrate the `ON DELTE CASCADE` referential action, delete records from the parent table where the `parent_id = 2`, which deletes all records in the parent table.

```
mysql> DELETE FROM parent WHERE id = 2;
```

Because all records in the child table are associated with `parent_id = 2`, the `ON DELETE CASCADE` referential action removes all records from the child table:

```
mysql> SELECT * FROM child;
Empty set (0.00 sec)
```

For more information about foreign key constraints, see Section 13.1.20.5, “FOREIGN KEY Constraints”.

3.6.7 Searching on Two Keys

An `OR` using a single key is well optimized, as is the handling of `AND`.

The one tricky case is that of searching on two different keys combined with `OR`:

```
SELECT field1_index, field2_index FROM test_table
WHERE field1_index = '1' OR  field2_index = '1'
```

This case is optimized. See Section 8.2.1.3, “Index Merge Optimization”.

You can also solve the problem efficiently by using a `UNION` that combines the output of two separate `SELECT` statements. See Section 13.2.18, “UNION Clause”.

Each `SELECT` searches only one key and can be optimized:

```
SELECT field1_index, field2_index
    FROM test_table WHERE field1_index = '1'
UNION
SELECT field1_index, field2_index
    FROM test_table WHERE field2_index = '1';
```

3.6.8 Calculating Visits Per Day

The following example shows how you can use the bit group functions to calculate the number of days per month a user has visited a Web page.

```
CREATE TABLE t1 (year YEAR, month INT UNSIGNED,
```

```
             day INT UNSIGNED);
INSERT INTO t1 VALUES(2000,1,1),(2000,1,20),(2000,1,30),(2000,2,2),
            (2000,2,23),(2000,2,23);
```

The example table contains year-month-day values representing visits by users to the page. To determine how many different days in each month these visits occur, use this query:

```
SELECT year,month,BIT_COUNT(BIT_OR(1<<day)) AS days FROM t1
       GROUP BY year,month;
```

Which returns:

```
+------+-------+------+
| year | month | days |
+------+-------+------+
| 2000 |     1 |    3 |
| 2000 |     2 |    2 |
+------+-------+------+
```

The query calculates how many different days appear in the table for each year/month combination, with automatic removal of duplicate entries.

3.6.9 Using AUTO_INCREMENT

The `AUTO_INCREMENT` attribute can be used to generate a unique identity for new rows:

```
CREATE TABLE animals (
     id MEDIUMINT NOT NULL AUTO_INCREMENT,
     name CHAR(30) NOT NULL,
     PRIMARY KEY (id)
);

INSERT INTO animals (name) VALUES
    ('dog'),('cat'),('penguin'),
    ('lax'),('whale'),('ostrich');

SELECT * FROM animals;
```

Which returns:

```
+----+---------+
| id | name    |
+----+---------+
|  1 | dog     |
|  2 | cat     |
|  3 | penguin |
|  4 | lax     |
|  5 | whale   |
|  6 | ostrich |
+----+---------+
```

No value was specified for the `AUTO_INCREMENT` column, so MySQL assigned sequence numbers automatically. You can also explicitly assign 0 to the column to generate sequence numbers, unless the `NO_AUTO_VALUE_ON_ZERO` SQL mode is enabled. For example:

```
INSERT INTO animals (id,name) VALUES(0,'groundhog');
```

If the column is declared `NOT NULL`, it is also possible to assign `NULL` to the column to generate sequence numbers. For example:

```
INSERT INTO animals (id,name) VALUES(NULL,'squirrel');
```

When you insert any other value into an `AUTO_INCREMENT` column, the column is set to that value and the sequence is reset so that the next automatically generated value follows sequentially from the largest column value. For example:

```
INSERT INTO animals (id,name) VALUES(100,'rabbit');
```

```
INSERT INTO animals (id,name) VALUES(NULL,'mouse');
SELECT * FROM animals;
+-----+-----------+
| id  | name      |
+-----+-----------+
|   1 | dog       |
|   2 | cat       |
|   3 | penguin   |
|   4 | lax       |
|   5 | whale     |
|   6 | ostrich   |
|   7 | groundhog |
|   8 | squirrel  |
| 100 | rabbit    |
| 101 | mouse     |
+-----+-----------+
```

Updating an existing `AUTO_INCREMENT` column value also resets the `AUTO_INCREMENT` sequence.

You can retrieve the most recent automatically generated `AUTO_INCREMENT` value with the `LAST_INSERT_ID()` SQL function or the `mysql_insert_id()` C API function. These functions are connection-specific, so their return values are not affected by another connection which is also performing inserts.

Use the smallest integer data type for the `AUTO_INCREMENT` column that is large enough to hold the maximum sequence value you require. When the column reaches the upper limit of the data type, the next attempt to generate a sequence number fails. Use the `UNSIGNED` attribute if possible to allow a greater range. For example, if you use `TINYINT`, the maximum permissible sequence number is 127. For `TINYINT UNSIGNED`, the maximum is 255. See Section 11.1.2, “Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT” for the ranges of all the integer types.

Note

For a multiple-row insert, `LAST_INSERT_ID()` and `mysql_insert_id()` actually return the `AUTO_INCREMENT` key from the *first* of the inserted rows. This enables multiple-row inserts to be reproduced correctly on other servers in a replication setup.

To start with an `AUTO_INCREMENT` value other than 1, set that value with `CREATE TABLE` or `ALTER TABLE`, like this:

```
mysql> ALTER TABLE tbl AUTO_INCREMENT = 100;
```

InnoDB Notes

For information about `AUTO_INCREMENT` usage specific to `InnoDB`, see Section 15.6.1.6, “AUTO_INCREMENT Handling in InnoDB”.

MyISAM Notes

- For `MyISAM` tables, you can specify `AUTO_INCREMENT` on a secondary column in a multiple-column index. In this case, the generated value for the `AUTO_INCREMENT` column is calculated as `MAX(auto_increment_column) + 1 WHERE prefix=given-prefix`. This is useful when you want to put data into ordered groups.

```
CREATE TABLE animals (
    grp ENUM('fish','mammal','bird') NOT NULL,
    id MEDIUMINT NOT NULL AUTO_INCREMENT,
    name CHAR(30) NOT NULL,
    PRIMARY KEY (grp,id)
) ENGINE=MyISAM;

INSERT INTO animals (grp,name) VALUES
    ('mammal','dog'),('mammal','cat'),
    ('bird','penguin'),('fish','lax'),('mammal','whale'),
```

```
    ('bird','ostrich');

SELECT * FROM animals ORDER BY grp,id;
```

Which returns:

```
+--------+----+---------+
| grp    | id | name    |
+--------+----+---------+
| fish   |  1 | lax     |
| mammal |  1 | dog     |
| mammal |  2 | cat     |
| mammal |  3 | whale   |
| bird   |  1 | penguin |
| bird   |  2 | ostrich |
+--------+----+---------+
```

In this case (when the `AUTO_INCREMENT` column is part of a multiple-column index), `AUTO_INCREMENT` values are reused if you delete the row with the biggest `AUTO_INCREMENT` value in any group. This happens even for `MyISAM` tables, for which `AUTO_INCREMENT` values normally are not reused.

- If the `AUTO_INCREMENT` column is part of multiple indexes, MySQL generates sequence values using the index that begins with the `AUTO_INCREMENT` column, if there is one. For example, if the `animals` table contained indexes `PRIMARY KEY (grp, id)` and `INDEX (id)`, MySQL would ignore the `PRIMARY KEY` for generating sequence values. As a result, the table would contain a single sequence, not a sequence per `grp` value.

Further Reading

More information about `AUTO_INCREMENT` is available here:

- How to assign the `AUTO_INCREMENT` attribute to a column: Section 13.1.20, “CREATE TABLE Statement”, and Section 13.1.9, “ALTER TABLE Statement”.
- How `AUTO_INCREMENT` behaves depending on the `NO_AUTO_VALUE_ON_ZERO` SQL mode: Section 5.1.11, “Server SQL Modes”.
- How to use the `LAST_INSERT_ID()` function to find the row that contains the most recent `AUTO_INCREMENT` value: Section 12.16, “Information Functions”.
- Setting the `AUTO_INCREMENT` value to be used: Section 5.1.8, “Server System Variables”.
- Section 15.6.1.6, “AUTO_INCREMENT Handling in InnoDB”
- `AUTO_INCREMENT` and replication: Section 17.5.1.1, “Replication and AUTO_INCREMENT”.
- Server-system variables related to `AUTO_INCREMENT` (`auto_increment_increment` and `auto_increment_offset`) that can be used for replication: Section 5.1.8, “Server System Variables”.

3.7 Using MySQL with Apache

There are programs that let you authenticate your users from a MySQL database and also let you write your log files into a MySQL table.

You can change the Apache logging format to be easily readable by MySQL by putting the following into the Apache configuration file:

```
LogFormat \
        "\"%h\",%{%Y%m%d%H%M%S}t,%>s,\"%b\",\"%{Content-Type}o\",  \
        \"%U\",\"%{Referer}i\",\"%{User-Agent}i\""
```

To load a log file in that format into MySQL, you can use a statement something like this:

```
LOAD DATA INFILE '/local/access_log' INTO TABLE tbl_name
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' ESCAPED BY '\\'
```

The named table should be created to have columns that correspond to those that the `LogFormat` line writes to the log file.

Chapter 4 MySQL Programs

Table of Contents

This chapter provides a brief overview of the MySQL command-line programs provided by Oracle Corporation. It also discusses the general syntax for specifying options when you run these programs. Most programs have options that are specific to their own operation, but the option syntax is similar for all of them. Finally, the chapter provides more detailed descriptions of individual programs, including which options they recognize.

4.1 Overview of MySQL Programs

There are many different programs in a MySQL installation. This section provides a brief overview of them. Later sections provide a more detailed description of each one, with the exception of NDB Cluster programs. Each program's description indicates its invocation syntax and the options that it supports. Section 23.5, “NDB Cluster Programs”, describes programs specific to NDB Cluster.

Most MySQL distributions include all of these programs, except for those programs that are platform-specific. (For example, the server startup scripts are not used on Windows.) The exception is that RPM distributions are more specialized. There is one RPM for the server, another for client programs, and so forth. If you appear to be missing one or more programs, see Chapter 2, *Installing and Upgrading MySQL*, for information on types of distributions and what they contain. It may be that you have a distribution that does not include all programs and you need to install an additional package.

Each MySQL program takes many different options. Most programs provide a `--help` option that you can use to get a description of the program's different options. For example, try `mysql --help`.

You can override default option values for MySQL programs by specifying options on the command line or in an option file. See Section 4.2, “Using MySQL Programs”, for general information on invoking programs and specifying program options.

The MySQL server, `mysqld`, is the main program that does most of the work in a MySQL installation. The server is accompanied by several related scripts that assist you in starting and stopping the server:

- `mysqld`

  The SQL daemon (that is, the MySQL server). To use client programs, `mysqld` must be running, because clients gain access to databases by connecting to the server. See Section 4.3.1, “mysqld — The MySQL Server”.

- `mysqld_safe`

  A server startup script. `mysqld_safe` attempts to start `mysqld`. See Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”.

- `mysql.server`

  A server startup script. This script is used on systems that use System V-style run directories containing scripts that start system services for particular run levels. It invokes `mysqld_safe` to start the MySQL server. See Section 4.3.3, “mysql.server — MySQL Server Startup Script”.

- `mysqld_multi`

  A server startup script that can start or stop multiple servers installed on the system. See Section 4.3.4, “mysqld_multi — Manage Multiple MySQL Servers”.

Several programs perform setup operations during MySQL installation or upgrading:

- `comp_err`

  This program is used during the MySQL build/installation process. It compiles error message files from the error source files. See Section 4.4.1, “comp_err — Compile MySQL Error Message File”.

- `mysql_secure_installation`

This program enables you to improve the security of your MySQL installation. See Section 4.4.2, “mysql_secure_installation — Improve MySQL Installation Security”.

- `mysql_ssl_rsa_setup`

  This program creates the SSL certificate and key files and RSA key-pair files required to support secure connections, if those files are missing. Files created by `mysql_ssl_rsa_setup` can be used for secure connections using SSL or RSA. See Section 4.4.3, “mysql_ssl_rsa_setup — Create SSL/RSA Files”.

- `mysql_tzinfo_to_sql`

  This program loads the time zone tables in the `mysql` database using the contents of the host system *zoneinfo* database (the set of files describing time zones). See Section 4.4.4, “mysql_tzinfo_to_sql — Load the Time Zone Tables”.

- `mysql_upgrade`

  Prior to MySQL 8.0.16, this program is used after a MySQL upgrade operation. It updates the grant tables with any changes that have been made in newer versions of MySQL, and checks tables for incompatibilities and repairs them if necessary. See Section 4.4.5, “mysql_upgrade — Check and Upgrade MySQL Tables”.

  As of MySQL 8.0.16, the MySQL server performs the upgrade tasks previously handled by `mysql_upgrade` (for details, see Section 2.10.3, “What the MySQL Upgrade Process Upgrades”).

MySQL client programs that connect to the MySQL server:

- `mysql`

  The command-line tool for interactively entering SQL statements or executing them from a file in batch mode. See Section 4.5.1, “mysql — The MySQL Command-Line Client”.

- `mysqladmin`

  A client that performs administrative operations, such as creating or dropping databases, reloading the grant tables, flushing tables to disk, and reopening log files. `mysqladmin` can also be used to retrieve version, process, and status information from the server. See Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”.

- `mysqlcheck`

  A table-maintenance client that checks, repairs, analyzes, and optimizes tables. See Section 4.5.3, “mysqlcheck — A Table Maintenance Program”.

- `mysqldump`

  A client that dumps a MySQL database into a file as SQL, text, or XML. See Section 4.5.4, “mysqldump — A Database Backup Program”.

- `mysqlimport`

  A client that imports text files into their respective tables using `LOAD DATA`. See Section 4.5.5, “mysqlimport — A Data Import Program”.

- `mysqlpump`

  A client that dumps a MySQL database into a file as SQL. See Section 4.5.6, “mysqlpump — A Database Backup Program”.

- `mysqlsh`

MySQL Shell is an advanced client and code editor for MySQL Server. See MySQL Shell 8.0. In addition to the provided SQL functionality, similar to `mysql`, MySQL Shell provides scripting capabilities for JavaScript and Python and includes APIs for working with MySQL. X DevAPI enables you to work with both relational and document data, see Chapter 20, *Using MySQL as a Document Store*. AdminAPI enables you to work with InnoDB Cluster, see MySQL AdminAPI.

- `mysqlshow`

  A client that displays information about databases, tables, columns, and indexes. See Section 4.5.7, "mysqlshow — Display Database, Table, and Column Information".

- `mysqlslap`

  A client that is designed to emulate client load for a MySQL server and report the timing of each stage. It works as if multiple clients are accessing the server. See Section 4.5.8, "mysqlslap — A Load Emulation Client".

MySQL administrative and utility programs:

- `innochecksum`

  An offline `InnoDB` offline file checksum utility. See Section 4.6.2, "innochecksum — Offline InnoDB File Checksum Utility".

- `myisam_ftdump`

  A utility that displays information about full-text indexes in `MyISAM` tables. See Section 4.6.3, "myisam_ftdump — Display Full-Text Index information".

- `myisamchk`

  A utility to describe, check, optimize, and repair `MyISAM` tables. See Section 4.6.4, "myisamchk — MyISAM Table-Maintenance Utility".

- `myisamlog`

  A utility that processes the contents of a `MyISAM` log file. See Section 4.6.5, "myisamlog — Display MyISAM Log File Contents".

- `myisampack`

  A utility that compresses `MyISAM` tables to produce smaller read-only tables. See Section 4.6.6, "myisampack — Generate Compressed, Read-Only MyISAM Tables".

- `mysql_config_editor`

  A utility that enables you to store authentication credentials in a secure, encrypted login path file named `.mylogin.cnf`. See Section 4.6.7, "mysql_config_editor — MySQL Configuration Utility".

- `mysql_migrate_keyring`

  A utility for migrating keys between one keyring component and another. See Section 4.6.8, "mysql_migrate_keyring — Keyring Key Migration Utility".

- `mysqlbinlog`

  A utility for reading statements from a binary log. The log of executed statements contained in the binary log files can be used to help recover from a crash. See Section 4.6.9, "mysqlbinlog — Utility for Processing Binary Log Files".

- `mysqldumpslow`

A utility to read and summarize the contents of a slow query log. See Section 4.6.10, “mysqldumpslow — Summarize Slow Query Log Files”.

MySQL program-development utilities:

- `mysql_config`

  A shell script that produces the option values needed when compiling MySQL programs. See Section 4.7.1, “mysql_config — Display Options for Compiling Clients”.

- `my_print_defaults`

  A utility that shows which options are present in option groups of option files. See Section 4.7.2, “my_print_defaults — Display Options from Option Files”.

Miscellaneous utilities:

- `lz4_decompress`

  A utility that decompresses `mysqlpump` output that was created using LZ4 compression. See Section 4.8.1, “lz4_decompress — Decompress mysqlpump LZ4-Compressed Output”.

- `perror`

  A utility that displays the meaning of system or MySQL error codes. See Section 4.8.2, “perror — Display MySQL Error Message Information”.

- `zlib_decompress`

  A utility that decompresses `mysqlpump` output that was created using ZLIB compression. See Section 4.8.3, “zlib_decompress — Decompress mysqlpump ZLIB-Compressed Output”.

Oracle Corporation also provides the MySQL Workbench GUI tool, which is used to administer MySQL servers and databases, to create, execute, and evaluate queries, and to migrate schemas and data from other relational database management systems for use with MySQL.

MySQL client programs that communicate with the server using the MySQL client/server library use the following environment variables.

Environment Variable	Meaning
`MYSQL_UNIX_PORT`	The default Unix socket file; used for connections to `localhost`
`MYSQL_TCP_PORT`	The default port number; used for TCP/IP connections
`MYSQL_DEBUG`	Debug trace options when debugging
`TMPDIR`	The directory where temporary tables and files are created

For a full list of environment variables used by MySQL programs, see Section 4.9, “Environment Variables”.

4.2 Using MySQL Programs

4.2.1 Invoking MySQL Programs

To invoke a MySQL program from the command line (that is, from your shell or command prompt), enter the program name followed by any options or other arguments needed to instruct the program what you want it to do. The following commands show some sample program invocations. `$>` represents the prompt for your command interpreter; it is not part of what you type. The particular

prompt you see depends on your command interpreter. Typical prompts are `$` for `sh`, `ksh`, or `bash`, `%` for `csh` or `tcsh`, and `C:\>` for the Windows `command.com` or `cmd.exe` command interpreters.

```
$> mysql --user=root test
$> mysqladmin extended-status variables
$> mysqlshow --help
$> mysqldump -u root personnel
```

Arguments that begin with a single or double dash (`-`, `--`) specify program options. Options typically indicate the type of connection a program should make to the server or affect its operational mode. Option syntax is described in Section 4.2.2, “Specifying Program Options”.

Nonoption arguments (arguments with no leading dash) provide additional information to the program. For example, the `mysql` program interprets the first nonoption argument as a database name, so the command `mysql --user=root test` indicates that you want to use the `test` database.

Later sections that describe individual programs indicate which options a program supports and describe the meaning of any additional nonoption arguments.

Some options are common to a number of programs. The most frequently used of these are the `--host` (or `-h`), `--user` (or `-u`), and `--password` (or `-p`) options that specify connection parameters. They indicate the host where the MySQL server is running, and the user name and password of your MySQL account. All MySQL client programs understand these options; they enable you to specify which server to connect to and the account to use on that server. Other connection options are `--port` (or `-P`) to specify a TCP/IP port number and `--socket` (or `-S`) to specify a Unix socket file on Unix (or named-pipe name on Windows). For more information on options that specify connection options, see Section 4.2.4, “Connecting to the MySQL Server Using Command Options”.

You may find it necessary to invoke MySQL programs using the path name to the `bin` directory in which they are installed. This is likely to be the case if you get a “program not found” error whenever you attempt to run a MySQL program from any directory other than the `bin` directory. To make it more convenient to use MySQL, you can add the path name of the `bin` directory to your `PATH` environment variable setting. That enables you to run a program by typing only its name, not its entire path name. For example, if `mysql` is installed in `/usr/local/mysql/bin`, you can run the program by invoking it as `mysql`, and it is not necessary to invoke it as `/usr/local/mysql/bin/mysql`.

Consult the documentation for your command interpreter for instructions on setting your `PATH` variable. The syntax for setting environment variables is interpreter-specific. (Some information is given in Section 4.2.9, “Setting Environment Variables”.) After modifying your `PATH` setting, open a new console window on Windows or log in again on Unix so that the setting goes into effect.

4.2.2 Specifying Program Options

There are several ways to specify options for MySQL programs:

- List the options on the command line following the program name. This is common for options that apply to a specific invocation of the program.
- List the options in an option file that the program reads when it starts. This is common for options that you want the program to use each time it runs.
- List the options in environment variables (see Section 4.2.9, “Setting Environment Variables”). This method is useful for options that you want to apply each time the program runs. In practice, option files are used more commonly for this purpose, but Section 5.8.3, “Running Multiple MySQL Instances on Unix”, discusses one situation in which environment variables can be very helpful. It describes a handy technique that uses such variables to specify the TCP/IP port number and Unix socket file for the server and for client programs.

Options are processed in order, so if an option is specified multiple times, the last occurrence takes precedence. The following command causes `mysql` to connect to the server running on `localhost`:

```
mysql -h example.com -h localhost
```

There is one exception: For `mysqld`, the *first* instance of the `--user` option is used as a security precaution, to prevent a user specified in an option file from being overridden on the command line.

If conflicting or related options are given, later options take precedence over earlier options. The following command runs `mysql` in “no column names” mode:

```
mysql --column-names --skip-column-names
```

MySQL programs determine which options are given first by examining environment variables, then by processing option files, and then by checking the command line. Because later options take precedence over earlier ones, the processing order means that environment variables have the lowest precedence and command-line options the highest.

For the server, one exception applies: The `mysqld-auto.cnf` option file in the data directory is processed last, so it takes precedence even over command-line options.

You can take advantage of the way that MySQL programs process options by specifying default option values for a program in an option file. That enables you to avoid typing them each time you run the program while enabling you to override the defaults if necessary by using command-line options.

4.2.2.1 Using Options on the Command Line

Program options specified on the command line follow these rules:

- Options are given after the command name.
- An option argument begins with one dash or two dashes, depending on whether it is a short form or long form of the option name. Many options have both short and long forms. For example, `-?` and `--help` are the short and long forms of the option that instructs a MySQL program to display its help message.
- Option names are case-sensitive. `-v` and `-V` are both legal and have different meanings. (They are the corresponding short forms of the `--verbose` and `--version` options.)
- Some options take a value following the option name. For example, `-h localhost` or `--host=localhost` indicate the MySQL server host to a client program. The option value tells the program the name of the host where the MySQL server is running.
- For a long option that takes a value, separate the option name and the value by an `=` sign. For a short option that takes a value, the option value can immediately follow the option letter, or there can be a space between: `-hlocalhost` and `-h localhost` are equivalent. An exception to this rule is the option for specifying your MySQL password. This option can be given in long form as `--password=`*`pass_val`* or as `--password`. In the latter case (with no password value given), the program interactively prompts you for the password. The password option also may be given in short form as `-p`*`pass_val`* or as `-p`. However, for the short form, if the password value is given, it must follow the option letter with *no intervening space*: If a space follows the option letter, the program has no way to tell whether a following argument is supposed to be the password value or some other kind of argument. Consequently, the following two commands have two completely different meanings:

  ```
  mysql -ptest
  mysql -p test
  ```

  The first command instructs `mysql` to use a password value of `test`, but specifies no default database. The second instructs `mysql` to prompt for the password value and to use `test` as the default database.
- Within option names, dash (`-`) and underscore (`_`) may be used interchangeably in most cases, although the leading dashes *cannot* be given as underscores. For example, `--skip-grant-tables` and `--skip_grant_tables` are equivalent.

In this Manual, we use dashes in option names, except where underscores are significant. This is the case with, for example, `--log-bin` and `--log_bin`, which are different options. We encourage you to do so as well.

- The MySQL server has certain command options that may be specified only at startup, and a set of system variables, some of which may be set at startup, at runtime, or both. System variable names use underscores rather than dashes, and when referenced at runtime (for example, using `SET` or `SELECT` statements), must be written using underscores:

```
SET GLOBAL general_log = ON;
SELECT @@GLOBAL.general_log;
```

  At server startup, the syntax for system variables is the same as for command options, so within variable names, dashes and underscores may be used interchangeably. For example, `--general_log=ON` and `--general-log=ON` are equivalent. (This is also true for system variables set within option files.)

- For options that take a numeric value, the value can be given with a suffix of `K`, `M`, or `G` to indicate a multiplier of 1024, 1024^2 or 1024^3. As of MySQL 8.0.14, a suffix can also be `T`, `P`, and `E` to indicate a multiplier of 1024^4, 1024^5 or 1024^6. Suffix letters can be uppercase or lowercase.

  For example, the following command tells `mysqladmin` to ping the server 1024 times, sleeping 10 seconds between each ping:

```
mysqladmin --count=1K --sleep=10 ping
```

- When specifying file names as option values, avoid the use of the `~` shell metacharacter. It might not be interpreted as you expect.

Option values that contain spaces must be quoted when given on the command line. For example, the `--execute` (or `-e`) option can be used with `mysql` to pass one or more semicolon-separated SQL statements to the server. When this option is used, `mysql` executes the statements in the option value and exits. The statements must be enclosed by quotation marks. For example:

```
$> mysql -u root -p -e "SELECT VERSION();SELECT NOW()"
Enter password: ******
+------------+
| VERSION()  |
+------------+
| 8.0.19     |
+------------+
+---------------------+
| NOW()               |
+---------------------+
| 2019-09-03 10:36:48 |
+---------------------+
$>
```

Note

The long form (`--execute`) is followed by an equal sign (`=`).

To use quoted values within a statement, you must either escape the inner quotation marks, or use a different type of quotation marks within the statement from those used to quote the statement itself. The capabilities of your command processor dictate your choices for whether you can use single or double quotation marks and the syntax for escaping quote characters. For example, if your command processor supports quoting with single or double quotation marks, you can use double quotation marks around the statement, and single quotation marks for any quoted values within the statement.

4.2.2.2 Using Option Files

Most MySQL programs can read startup options from option files (sometimes called configuration files). Option files provide a convenient way to specify commonly used options so that they need not be entered on the command line each time you run a program.

To determine whether a program reads option files, invoke it with the `--help` option. (For `mysqld`, use `--verbose` and `--help`.) If the program reads option files, the help message indicates which files it looks for and which option groups it recognizes.

Note

A MySQL program started with the `--no-defaults` option reads no option files other than `.mylogin.cnf`.

A server started with the `persisted_globals_load` system variable disabled does not read `mysqld-auto.cnf`.

Many option files are plain text files, created using any text editor. The exceptions are:

- The `.mylogin.cnf` file that contains login path options. This is an encrypted file created by the `mysql_config_editor` utility. See Section 4.6.7, “mysql_config_editor — MySQL Configuration Utility”. A “login path” is an option group that permits only certain options: `host`, `user`, `password`, `port` and `socket`. Client programs specify which login path to read from `.mylogin.cnf` using the `--login-path` option.

  To specify an alternative login path file name, set the `MYSQL_TEST_LOGIN_FILE` environment variable. This variable is used by the `mysql-test-run.pl` testing utility, but also is recognized by `mysql_config_editor` and by MySQL clients such as `mysql`, `mysqladmin`, and so forth.

- The `mysqld-auto.cnf` file in the data directory. This JSON-format file contains persisted system variable settings. It is created by the server upon execution of `SET PERSIST` or `SET PERSIST_ONLY` statements. See Section 5.1.9.3, “Persisted System Variables”. Management of `mysqld-auto.cnf` should be left to the server and not performed manually.

- Option File Processing Order

- Option File Syntax

- Option File Inclusions

Option File Processing Order

MySQL looks for option files in the order described in the following discussion and reads any that exist. If an option file you want to use does not exist, create it using the appropriate method, as just discussed.

Note

For information about option files used with NDB Cluster programs, see Section 23.4, “Configuration of NDB Cluster”.

On Windows, MySQL programs read startup options from the files shown in the following table, in the specified order (files listed first are read first, files read later take precedence).

Table 4.1 Option Files Read on Windows Systems

File Name	Purpose
`%WINDIR%\my.ini`, `%WINDIR%\my.cnf`	Global options
`C:\my.ini`, `C:\my.cnf`	Global options
`BASEDIR``\my.ini`, *`BASEDIR`*`\my.cnf`	Global options

File Name	Purpose
`defaults-extra-file`	The file specified with `--defaults-extra-file`, if any
`%APPDATA%\MySQL\.mylogin.cnf`	Login path options (clients only)
`DATADIR``\mysqld-auto.cnf`	System variables persisted with `SET PERSIST` or `SET PERSIST_ONLY` (server only)

In the preceding table, `%WINDIR%` represents the location of your Windows directory. This is commonly `C:\WINDOWS`. Use the following command to determine its exact location from the value of the `WINDIR` environment variable:

```
C:\> echo %WINDIR%
```

`%APPDATA%` represents the value of the Windows application data directory. Use the following command to determine its exact location from the value of the `APPDATA` environment variable:

```
C:\> echo %APPDATA%
```

`BASEDIR` represents the MySQL base installation directory. When MySQL 8.0 has been installed using MySQL Installer, this is typically `C:\`*`PROGRAMDIR`*`\MySQL\MySQL Server 8.0` in which *`PROGRAMDIR`* represents the programs directory (usually `Program Files` for English-language versions of Windows). See Section 2.3.3, “MySQL Installer for Windows”.

Important

Although MySQL Installer places most files under *`PROGRAMDIR`*, it installs `my.ini` under the `C:\ProgramData\MySQL\MySQL Server 8.0\` directory by default.

`DATADIR` represents the MySQL data directory. As used to find `mysqld-auto.cnf`, its default value is the data directory location built in when MySQL was compiled, but can be changed by `--datadir` specified as an option-file or command-line option processed before `mysqld-auto.cnf` is processed.

On Unix and Unix-like systems, MySQL programs read startup options from the files shown in the following table, in the specified order (files listed first are read first, files read later take precedence).

Note

On Unix platforms, MySQL ignores configuration files that are world-writable. This is intentional as a security measure.

Table 4.2 Option Files Read on Unix and Unix-Like Systems

File Name	Purpose
`/etc/my.cnf`	Global options
`/etc/mysql/my.cnf`	Global options
`SYSCONFDIR``/my.cnf`	Global options
`$MYSQL_HOME/my.cnf`	Server-specific options (server only)
`defaults-extra-file`	The file specified with `--defaults-extra-file`, if any
`~/.my.cnf`	User-specific options
`~/.mylogin.cnf`	User-specific login path options (clients only)
`DATADIR``/mysqld-auto.cnf`	System variables persisted with `SET PERSIST` or `SET PERSIST_ONLY` (server only)

In the preceding table, ~ represents the current user's home directory (the value of $HOME).

SYSCONFDIR represents the directory specified with the SYSCONFDIR option to CMake when MySQL was built. By default, this is the etc directory located under the compiled-in installation directory.

MYSQL_HOME is an environment variable containing the path to the directory in which the server-specific my.cnf file resides. If MYSQL_HOME is not set and you start the server using the mysqld_safe program, mysqld_safe sets it to *BASEDIR*, the MySQL base installation directory.

DATADIR represents the MySQL data directory. As used to find mysqld-auto.cnf, its default value is the data directory location built in when MySQL was compiled, but can be changed by --datadir specified as an option-file or command-line option processed before mysqld-auto.cnf is processed.

If multiple instances of a given option are found, the last instance takes precedence, with one exception: For mysqld, the *first* instance of the --user option is used as a security precaution, to prevent a user specified in an option file from being overridden on the command line.

Option File Syntax

The following description of option file syntax applies to files that you edit manually. This excludes .mylogin.cnf, which is created using mysql_config_editor and is encrypted, and mysqld-auto.cnf, which the server creates in JSON format.

Any long option that may be given on the command line when running a MySQL program can be given in an option file as well. To get the list of available options for a program, run it with the --help option. (For mysqld, use --verbose and --help.)

The syntax for specifying options in an option file is similar to command-line syntax (see Section 4.2.2.1, "Using Options on the Command Line"). However, in an option file, you omit the leading two dashes from the option name and you specify only one option per line. For example, --quick and --host=localhost on the command line should be specified as quick and host=localhost on separate lines in an option file. To specify an option of the form --loose-*opt_name* in an option file, write it as loose-*opt_name*.

Empty lines in option files are ignored. Nonempty lines can take any of the following forms:

- #*comment*, ;*comment*

  Comment lines start with # or ;. A # comment can start in the middle of a line as well.

- [*group*]

  group is the name of the program or group for which you want to set options. After a group line, any option-setting lines apply to the named group until the end of the option file or another group line is given. Option group names are not case-sensitive.

- *opt_name*

  This is equivalent to --*opt_name* on the command line.

- *opt_name*=*value*

  This is equivalent to --*opt_name*=*value* on the command line. In an option file, you can have spaces around the = character, something that is not true on the command line. The value optionally can be enclosed within single quotation marks or double quotation marks, which is useful if the value contains a # comment character.

Leading and trailing spaces are automatically deleted from option names and values.

You can use the escape sequences \b, \t, \n, \r, \\, and \s in option values to represent the backspace, tab, newline, carriage return, backslash, and space characters. In option files, these escaping rules apply:

- A backslash followed by a valid escape sequence character is converted to the character represented by the sequence. For example, `\s` is converted to a space.
- A backslash not followed by a valid escape sequence character remains unchanged. For example, `\S` is retained as is.

The preceding rules mean that a literal backslash can be given as `\\`, or as `\` if it is not followed by a valid escape sequence character.

The rules for escape sequences in option files differ slightly from the rules for escape sequences in string literals in SQL statements. In the latter context, if "*x*" is not a valid escape sequence character, `\`*x* becomes "*x*" rather than `\`*x*. See Section 9.1.1, "String Literals".

The escaping rules for option file values are especially pertinent for Windows path names, which use `\` as a path name separator. A separator in a Windows path name must be written as `\\` if it is followed by an escape sequence character. It can be written as `\\` or `\` if it is not. Alternatively, `/` may be used in Windows path names and is treated as `\`. Suppose that you want to specify a base directory of `C:\Program Files\MySQL\MySQL Server 8.0` in an option file. This can be done several ways. Some examples:

```
basedir="C:\Program Files\MySQL\MySQL Server 8.0"
basedir="C:\\Program Files\\MySQL\\MySQL Server 8.0"
basedir="C:/Program Files/MySQL/MySQL Server 8.0"
basedir=C:\\Program\sFiles\\MySQL\\MySQL\sServer\s8.0
```

If an option group name is the same as a program name, options in the group apply specifically to that program. For example, the `[mysqld]` and `[mysql]` groups apply to the `mysqld` server and the `mysql` client program, respectively.

The `[client]` option group is read by all client programs provided in MySQL distributions (but *not* by `mysqld`). To understand how third-party client programs that use the C API can use option files, see the C API documentation at mysql_options().

The `[client]` group enables you to specify options that apply to all clients. For example, `[client]` is the appropriate group to use to specify the password for connecting to the server. (But make sure that the option file is accessible only by yourself, so that other people cannot discover your password.) Be sure not to put an option in the `[client]` group unless it is recognized by *all* client programs that you use. Programs that do not understand the option quit after displaying an error message if you try to run them.

List more general option groups first and more specific groups later. For example, a `[client]` group is more general because it is read by all client programs, whereas a `[mysqldump]` group is read only by `mysqldump`. Options specified later override options specified earlier, so putting the option groups in the order `[client]`, `[mysqldump]` enables `mysqldump`-specific options to override `[client]` options.

Here is a typical global option file:

```
[client]
port=3306
socket=/tmp/mysql.sock

[mysqld]
port=3306
socket=/tmp/mysql.sock
key_buffer_size=16M
max_allowed_packet=128M

[mysqldump]
quick
```

Here is a typical user option file:

```
[client]
```

```
# The following password is sent to all standard MySQL clients
password="my password"

[mysql]
no-auto-rehash
connect_timeout=2
```

To create option groups to be read only by `mysqld` servers from specific MySQL release series, use groups with names of `[mysqld-5.7]`, `[mysqld-8.0]`, and so forth. The following group indicates that the `sql_mode` setting should be used only by MySQL servers with 8.0.x version numbers:

```
[mysqld-8.0]
sql_mode=TRADITIONAL
```

Option File Inclusions

It is possible to use `!include` directives in option files to include other option files and `!includedir` to search specific directories for option files. For example, to include the `/home/mydir/myopt.cnf` file, use the following directive:

```
!include /home/mydir/myopt.cnf
```

To search the `/home/mydir` directory and read option files found there, use this directive:

```
!includedir /home/mydir
```

MySQL makes no guarantee about the order in which option files in the directory are read.

Note

Any files to be found and included using the `!includedir` directive on Unix operating systems *must* have file names ending in `.cnf`. On Windows, this directive checks for files with the `.ini` or `.cnf` extension.

Write the contents of an included option file like any other option file. That is, it should contain groups of options, each preceded by a `[group]` line that indicates the program to which the options apply.

While an included file is being processed, only those options in groups that the current program is looking for are used. Other groups are ignored. Suppose that a `my.cnf` file contains this line:

```
!include /home/mydir/myopt.cnf
```

And suppose that `/home/mydir/myopt.cnf` looks like this:

```
[mysqladmin]
force

[mysqld]
key_buffer_size=16M
```

If `my.cnf` is processed by `mysqld`, only the `[mysqld]` group in `/home/mydir/myopt.cnf` is used. If the file is processed by `mysqladmin`, only the `[mysqladmin]` group is used. If the file is processed by any other program, no options in `/home/mydir/myopt.cnf` are used.

The `!includedir` directive is processed similarly except that all option files in the named directory are read.

If an option file contains `!include` or `!includedir` directives, files named by those directives are processed whenever the option file is processed, no matter where they appear in the file.

For inclusion directives to work, the file path should not be specified within quotes and should have no escape sequences. For example, the following statements provided in `my.ini` read the option file `myopts.ini`:

```
!include C:/ProgramData/MySQL/MySQL Server/myopts.ini
!include C:\ProgramData\MySQL\MySQL Server\myopts.ini
!include C:\\ProgramData\\MySQL\\MySQL Server\\myopts.ini
```

On Windows, if `!include /path/to/extra.ini` is the last line in the file, make sure that a newline is appended at the end; otherwise, the line is ignored.

4.2.2.3 Command-Line Options that Affect Option-File Handling

Most MySQL programs that support option files handle the following options. Because these options affect option-file handling, they must be given on the command line and not in an option file. To work properly, each of these options must be given before other options, with these exceptions:

- `--print-defaults` may be used immediately after `--defaults-file`, `--defaults-extra-file`, or `--login-path`.
- On Windows, if the server is started with the `--defaults-file` and `--install` options, `--install` must be first. See Section 2.3.4.8, “Starting MySQL as a Windows Service”.

When specifying file names as option values, avoid the use of the `~` shell metacharacter because it might not be interpreted as you expect.

- `--defaults-extra-file=file_name`

  Read this option file after the global option file but (on Unix) before the user option file and (on all platforms) before the login path file. (For information about the order in which option files are used, see Section 4.2.2.2, “Using Option Files”.) If the file does not exist or is otherwise inaccessible, an error occurs. If *`file_name`* is not an absolute path name, it is interpreted relative to the current directory.

  See the introduction to this section regarding constraints on the position in which this option may be specified.

- `--defaults-file=file_name`

  Read only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  Exceptions: Even with `--defaults-file`, `mysqld` reads `mysqld-auto.cnf` and client programs read `.mylogin.cnf`.

  See the introduction to this section regarding constraints on the position in which this option may be specified.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, the `mysql` client normally reads the `[client]` and `[mysql]` groups. If this option is given as `--defaults-group-suffix=_other`, `mysql` also reads the `[client_other]` and `[mysql_other]` groups.

- `--login-path=name`

  Read options from the named login path in the `.mylogin.cnf` login path file. A “login path” is an option group containing options that specify which MySQL server to connect to and which account to authenticate as. To create or modify a login path file, use the `mysql_config_editor` utility. See Section 4.6.7, “mysql_config_editor — MySQL Configuration Utility”.

  A client program reads the option group corresponding to the named login path, in addition to option groups that the program reads by default. Consider this command:

```
mysql --login-path=mypath
```

By default, the `mysql` client reads the `[client]` and `[mysql]` option groups. So for the command shown, `mysql` reads `[client]` and `[mysql]` from other option files, and `[client]`, `[mysql]`, and `[mypath]` from the login path file.

Client programs read the login path file even when the `--no-defaults` option is used.

To specify an alternate login path file name, set the `MYSQL_TEST_LOGIN_FILE` environment variable.

See the introduction to this section regarding constraints on the position in which this option may be specified.

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

  The exception is that client programs read the `.mylogin.cnf` login path file, if it exists, even when `--no-defaults` is used. This permits passwords to be specified in a safer way than on the command line even if `--no-defaults` is present. To create `.mylogin.cnf`, use the `mysql_config_editor` utility. See Section 4.6.7, “mysql_config_editor — MySQL Configuration Utility”.

- `--print-defaults`

  Print the program name and all options that it gets from option files. Password values are masked.

  See the introduction to this section regarding constraints on the position in which this option may be specified.

4.2.2.4 Program Option Modifiers

Some options are “boolean” and control behavior that can be turned on or off. For example, the `mysql` client supports a `--column-names` option that determines whether or not to display a row of column names at the beginning of query results. By default, this option is enabled. However, you may want to disable it in some instances, such as when sending the output of `mysql` into another program that expects to see only data and not an initial header line.

To disable column names, you can specify the option using any of these forms:

```
--disable-column-names
--skip-column-names
--column-names=0
```

The `--disable` and `--skip` prefixes and the `=0` suffix all have the same effect: They turn the option off.

The “enabled” form of the option may be specified in any of these ways:

```
--column-names
--enable-column-names
--column-names=1
```

The values `ON`, `TRUE`, `OFF`, and `FALSE` are also recognized for boolean options (not case-sensitive).

If an option is prefixed by `--loose`, a program does not exit with an error if it does not recognize the option, but instead issues only a warning:

```
$> mysql --loose-no-such-option
```

```
mysql: WARNING: unknown option '--loose-no-such-option'
```

The `--loose` prefix can be useful when you run programs from multiple installations of MySQL on the same machine and list options in an option file. An option that may not be recognized by all versions of a program can be given using the `--loose` prefix (or `loose` in an option file). Versions of the program that recognize the option process it normally, and versions that do not recognize it issue a warning and ignore it.

The `--maximum` prefix is available for `mysqld` only and permits a limit to be placed on how large client programs can set session system variables. To do this, use a `--maximum` prefix with the variable name. For example, `--maximum-max_heap_table_size=32M` prevents any client from making the heap table size limit larger than 32M.

The `--maximum` prefix is intended for use with system variables that have a session value. If applied to a system variable that has only a global value, an error occurs. For example, with `--maximum-back_log=200`, the server produces this error:

```
Maximum value of 'back_log' cannot be set
```

4.2.2.5 Using Options to Set Program Variables

Many MySQL programs have internal variables that can be set at runtime using the `SET` statement. See Section 13.7.6.1, “SET Syntax for Variable Assignment”, and Section 5.1.9, “Using System Variables”.

Most of these program variables also can be set at server startup by using the same syntax that applies to specifying program options. For example, `mysql` has a `max_allowed_packet` variable that controls the maximum size of its communication buffer. To set the `max_allowed_packet` variable for `mysql` to a value of 16MB, use either of the following commands:

```
mysql --max_allowed_packet=16777216
mysql --max_allowed_packet=16M
```

The first command specifies the value in bytes. The second specifies the value in megabytes. For variables that take a numeric value, the value can be given with a suffix of `K`, `M`, or `G` to indicate a multiplier of 1024, 1024^2 or 1024^3. (For example, when used to set `max_allowed_packet`, the suffixes indicate units of kilobytes, megabytes, or gigabytes.) As of MySQL 8.0.14, a suffix can also be `T`, `P`, and `E` to indicate a multiplier of 1024^4, 1024^5 or 1024^6. Suffix letters can be uppercase or lowercase.

In an option file, variable settings are given without the leading dashes:

```
[mysql]
max_allowed_packet=16777216
```

Or:

```
[mysql]
max_allowed_packet=16M
```

If you like, underscores in an option name can be specified as dashes. The following option groups are equivalent. Both set the size of the server's key buffer to 512MB:

```
[mysqld]
key_buffer_size=512M

[mysqld]
key-buffer-size=512M
```

Suffixes for specifying a value multiplier can be used when setting a variable at program invocation time, but not to set the value with `SET` at runtime. On the other hand, with `SET`, you can assign a

variable's value using an expression, which is not true when you set a variable at server startup. For example, the first of the following lines is legal at program invocation time, but the second is not:

```
$> mysql --max_allowed_packet=16M
$> mysql --max_allowed_packet=16*1024*1024
```

Conversely, the second of the following lines is legal at runtime, but the first is not:

```
mysql> SET GLOBAL max_allowed_packet=16M;
mysql> SET GLOBAL max_allowed_packet=16*1024*1024;
```

4.2.2.6 Option Defaults, Options Expecting Values, and the = Sign

By convention, long forms of options that assign a value are written with an equals (=) sign, like this:

```
mysql --host=tonfisk --user=jon
```

For options that require a value (that is, not having a default value), the equal sign is not required, and so the following is also valid:

```
mysql --host tonfisk --user jon
```

In both cases, the `mysql` client attempts to connect to a MySQL server running on the host named “tonfisk” using an account with the user name “jon”.

Due to this behavior, problems can occasionally arise when no value is provided for an option that expects one. Consider the following example, where a user connects to a MySQL server running on host `tonfisk` as user `jon`:

```
$> mysql --host 85.224.35.45 --user jon
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 8.0.32 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT CURRENT_USER();
+----------------+
| CURRENT_USER() |
+----------------+
| jon@%          |
+----------------+
1 row in set (0.00 sec)
```

Omitting the required value for one of these option yields an error, such as the one shown here:

```
$> mysql --host 85.224.35.45 --user
mysql: option '--user' requires an argument
```

In this case, `mysql` was unable to find a value following the `--user` option because nothing came after it on the command line. However, if you omit the value for an option that is *not* the last option to be used, you obtain a different error that you may not be expecting:

```
$> mysql --host --user jon
ERROR 2005 (HY000): Unknown MySQL server host '--user' (1)
```

Because `mysql` assumes that any string following `--host` on the command line is a host name, `--host --user` is interpreted as `--host=--user`, and the client attempts to connect to a MySQL server running on a host named “--user”.

Options having default values always require an equal sign when assigning a value; failing to do so causes an error. For example, the MySQL server `--log-error` option has the default value *`host_name`*`.err`, where *`host_name`* is the name of the host on which MySQL is running. Assume

that you are running MySQL on a computer whose host name is “tonfisk”, and consider the following invocation of `mysqld_safe`:

```
$> mysqld_safe &
[1] 11699
$> 080112 12:53:40 mysqld_safe Logging to '/usr/local/mysql/var/tonfisk.err'.
080112 12:53:40 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var
$>
```

After shutting down the server, restart it as follows:

```
$> mysqld_safe --log-error &
[1] 11699
$> 080112 12:53:40 mysqld_safe Logging to '/usr/local/mysql/var/tonfisk.err'.
080112 12:53:40 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var
$>
```

The result is the same, since `--log-error` is not followed by anything else on the command line, and it supplies its own default value. (The `&` character tells the operating system to run MySQL in the background; it is ignored by MySQL itself.) Now suppose that you wish to log errors to a file named `my-errors.err`. You might try starting the server with `--log-error my-errors`, but this does not have the intended effect, as shown here:

```
$> mysqld_safe --log-error my-errors &
[1] 31357
$> 080111 22:53:31 mysqld_safe Logging to '/usr/local/mysql/var/tonfisk.err'.
080111 22:53:32 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var
080111 22:53:34 mysqld_safe mysqld from pid file /usr/local/mysql/var/tonfisk.pid ended

[1]+  Done                    ./mysqld_safe --log-error my-errors
```

The server attempted to start using `/usr/local/mysql/var/tonfisk.err` as the error log, but then shut down. Examining the last few lines of this file shows the reason:

```
$> tail /usr/local/mysql/var/tonfisk.err
2013-09-24T15:36:22.278034Z 0 [ERROR] Too many arguments (first extra is 'my-errors').
2013-09-24T15:36:22.278059Z 0 [Note] Use --verbose --help to get a list of available options!
2013-09-24T15:36:22.278076Z 0 [ERROR] Aborting
2013-09-24T15:36:22.279704Z 0 [Note] InnoDB: Starting shutdown...
2013-09-24T15:36:23.777471Z 0 [Note] InnoDB: Shutdown completed; log sequence number 2319086
2013-09-24T15:36:23.780134Z 0 [Note] mysqld: Shutdown complete
```

Because the `--log-error` option supplies a default value, you must use an equal sign to assign a different value to it, as shown here:

```
$> mysqld_safe --log-error=my-errors &
[1] 31437
$> 080111 22:54:15 mysqld_safe Logging to '/usr/local/mysql/var/my-errors.err'.
080111 22:54:15 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var

$>
```

Now the server has been started successfully, and is logging errors to the file `/usr/local/mysql/var/my-errors.err`.

Similar issues can arise when specifying option values in option files. For example, consider a `my.cnf` file that contains the following:

```
[mysql]

host
user
```

When the `mysql` client reads this file, these entries are parsed as `--host --user` or `--host=--user`, with the result shown here:

```
$> mysql
```

```
ERROR 2005 (HY000): Unknown MySQL server host '--user' (1)
```

However, in option files, an equal sign is not assumed. Suppose the `my.cnf` file is as shown here:

```
[mysql]

user jon
```

Trying to start `mysql` in this case causes a different error:

```
$> mysql
mysql: unknown option '--user jon'
```

A similar error would occur if you were to write `host tonfisk` in the option file rather than `host=tonfisk`. Instead, you must use the equal sign:

```
[mysql]

user=jon
```

Now the login attempt succeeds:

```
$> mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5
Server version: 8.0.32 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT USER();
+---------------+
| USER()        |
+---------------+
| jon@localhost |
+---------------+
1 row in set (0.00 sec)
```

This is not the same behavior as with the command line, where the equal sign is not required:

```
$> mysql --user jon --host tonfisk
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 8.0.32 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT USER();
+---------------+
| USER()        |
+---------------+
| jon@tonfisk   |
+---------------+
1 row in set (0.00 sec)
```

Specifying an option requiring a value without a value in an option file causes the server to abort with an error.

4.2.3 Command Options for Connecting to the Server

This section describes options supported by most MySQL client programs that control how client programs establish connections to the server, whether connections are encrypted, and whether connections are compressed. These options can be given on the command line or in an option file.

- Command Options for Connection Establishment
- Command Options for Encrypted Connections
- Command Options for Connection Compression

Command Options for Connection Establishment

This section describes options that control how client programs establish connections to the server. For additional information and examples showing how to use them, see Section 4.2.4, “Connecting to the MySQL Server Using Command Options”.

Table 4.3 Connection-Establishment Option Summary

Option Name	Description	Introduced
--default-auth	Authentication plugin to use	
--host	Host on which MySQL server is located	
--password	Password to use when connecting to server	
--password1	First multifactor authentication password to use when connecting to server	8.0.27
--password2	Second multifactor authentication password to use when connecting to server	8.0.27
--password3	Third multifactor authentication password to use when connecting to server	8.0.27
--pipe	Connect to server using named pipe (Windows only)	
--plugin-dir	Directory where plugins are installed	
--port	TCP/IP port number for connection	
--protocol	Transport protocol to use	
--shared-memory-base-name	Shared-memory name for shared-memory connections (Windows only)	
--socket	Unix socket file or Windows named pipe to use	
--user	MySQL user name to use when connecting to server	

- `--default-auth=`*`plugin`*

  A hint about which client-side authentication plugin to use. See Section 6.2.17, “Pluggable Authentication”.

- `--host=`*`host_name`*, `-h` *`host_name`*

  The host on which the MySQL server is running. The value can be a host name, IPv4 address, or IPv6 address. The default value is `localhost`.

- `--password[=`*`pass_val`*`]`, `-p[`*`pass_val`*`]`

  The password of the MySQL account used for connecting to the server. The password value is optional. If not given, the client program prompts for one. If given, there must be *no space* between `--password=` or `-p` and the password following it. If no password option is specified, the default is to send no password.