Karthik Appigatla

MySQL 8 Cookbook

Over 150 recipes for high-performance database querying and administration

Packt

MySQL 8 Cookbook

Over 150 recipes for high-performance database querying and administration

Karthik Appigatla

Packt

BIRMINGHAM - MUMBAI

MySQL 8 Cookbook

Copyright © 2018 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

Commissioning Editor: Amey Varangaokar
Acquisition Editor: Viraj Madhav
Content Development Editors: Aishwarya Pandere
Technical Editor: Prasad Ramesh
Copy Editor: Vikrant Phadkay
Project Coordinator: Nidhi Joshi
Proofreader: Safis Editing
Indexer: Pratik Shirodkar
Graphics: Tania Dutta
Production Coordinator: Shraddha Falebhai

First published: January 2018
Production reference: 1230118

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.

ISBN 978-1-78839-580-9

www.packtpub.com

To my mother, A.S Gayathri, and to the memory of my father, A.S.R.V.S.N Murty, for their sacrifices and for exemplifying the power of determination.

– Karthik Appigatla

mapt.io

Mapt is an online digital library that gives you full access to over 5,000 books and videos, as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

Why subscribe?

- Spend less time learning and more time coding with practical eBooks and Videos from over 4,000 industry professionals
- Improve your learning with Skill Plans built especially for you
- Get a free eBook or video every month
- Mapt is fully searchable
- Copy and paste, print, and bookmark content

PacktPub.com

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

Contributors

About the author

Karthik Appigatla is a highly reputed database architect and is very famous for performance tuning. He has been consulted by many companies all over the world for designing, performance tuning, building database infrastructure, and training. In his decade of experience, he has worked for companies such as Yahoo, Pythian, and Percona. Currently, he is working for LinkedIn, where he has innovated a new way of analyzing queries. He gave a talk about this at SRECon, Dublin in 2017.

I would like to acknowledge the encouragement from my wife, Lalitha, and my brother, Kashyap. This book would not have completed without the cooperation of my little daughter, Samhita.

About the reviewers

Marco Ippolito is an Italian software engineer working as Director of Software Development for Imagining IT. Marco completed his postgraduate in Software Engineering in Oxford and has worked for large corporations such as Intel, HP, Google, Dell, and Oracle (in the acquired MySQL team), as well as for start-ups such as `@platformsh` and for large production users of MySQL such as `@bookingcom` (Twitter). He can be reached at `marco.ippolito@imaginingit.com` and has experience working in teams speaking Italian, English, Spanish, Brazilian Portuguese, German, and French, remotely or onsite.

Kedar Mohaniraj Vaijanapurkar is a MySQL database consultant with over a decade of experience, ranging from programming to database administration. He aims to spread happiness with the MySQL database systems he works on. Apart from working with MySQL and related open source technologies, he also explores cloud, automation, and NoSQL. He is a 5-year Pythian laureate living with his awesome family in the cultured city of Vadodara, India. You can reach him at `kedar@nitty-witty.com`.

Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit authors.packtpub.com and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Table of Contents

Preface
- Who this book is for
- What this book covers
- To get the most out of this book
  - Conventions used
- Sections
  - Getting ready
  - How to do it…
  - How it works…
  - There's more…
  - See also
- Get in touch
  - Reviews

1. MySQL 8 - Installing and Upgrading
   - Introduction
   - Installing MySQL using YUM/APT
     - How to do it...
       - Using YUM repositories
       - Using APT repositories
   - Installing MySQL 8.0 using RPM or DEB files
     - How to do it...
       - Using the RPM bundle
       - Using the APT bundle
   - Installing MySQL on Linux using Generic Binaries
     - How to do it...
     - There's more...
   - Starting or Stopping MySQL 8
     - How to do it...
       - Starting the MySQL 8.0 server
       - Stopping the MySQL 8.0 server

            - Checking the status of the MySQL 8.0 server
- Uninstalling MySQL 8
    - How to do it...
        - On YUM-based systems
        - On APT-based systems
        - Uninstalling Binaries
- Managing the MySQL Server with systemd
    - How to do it...
- Downgrading from MySQL 8.0
    - How to do it...
        - In-place Downgrades
            - Using YUM repositories
            - Using APT Repositories
            - Using the RPM or APT bundle
            - Using Generic Binaries
        - Logical Downgrades
            - Using YUM Repositories
            - Using APT Repositories
            - Using RPM or APT bundles
            - Using Generic Binaries
- Upgrading to MySQL 8.0
    - Getting ready
    - How to do it...
        - In-place upgrades
            - YUM-based systems
            - APT-based systems
            - Using RPM or APT bundles
            - Using Generic Binaries
        - Logical Upgrades
- Installing MySQL utilities
    - How to do it...
        - On YUM-based systems
        - On APT-based systems

2. Using MySQL
- Introduction
- Connecting to MySQL using the command-line client
  - Getting ready
  - How to do it...
  - See also
- Creating databases
  - How to do it...
  - See also
- Creating tables
  - How to do it...
    - Cloning table structure
  - See also
- Inserting, updating, and deleting rows
  - How to do it...
    - Inserting
    - Updating
    - Deleting
    - REPLACE, INSERT, ON DUPLICATE KEY UPDATE
    - Truncating tables
- Loading sample data
  - How to do it...
- Selecting data
  - How to do it...
    - Selecting columns
    - Count
    - Filter based on condition
    - Operators
    - Simple pattern matching
    - Regular expressions
    - Limiting results
    - Using the table alias
- Sorting results

        How to do it...
    Grouping results (aggregate functions)
        How to do it...
            COUNT
            SUM
            AVERAGE
            DISTINCT
            Filtering using HAVING
        See also
    Creating users
        How to do it...
        See also
    Granting and revoking access to users
        How to do it...
            Granting privileges
            Checking grants
            Revoking grants
            Modifying the mysql.user table
            Setting password expiry for users
            Locking users
            Creating roles for users
    Selecting data into a file and table
        How to do it...
            Saving as a file
            Saving as a table
    Loading data into a table
        How to do it...
    Joining tables
        How to do it...
            Identifying Duplicates using SELF JOIN
            Using SUB queries
            Finding mismatched rows between tables
    Stored procedures

- How to do it...
- There's more...
- See also
- Functions
  - How to do it...
    - Inbuilt functions
  - See also
- Triggers
  - How to do it...
  - See also
- Views
  - How to do it...
- Events
  - How to do it...
    - Access control
  - See also
- Getting information about databases and tables
  - How to do it...
    - TABLES
    - COLUMNS
    - FILES
    - INNODB_SYS_TABLESPACES
    - INNODB_TABLESTATS
    - PROCESSLIST
  - See also

3. Using MySQL (Advanced)
  - Introduction
  - Using JSON
    - How to do it...
      - Insert JSON
      - Retrieve JSON
      - JSON functions
        - Pretty view

            - Searching
            - Modifying
            - Removing
            - Other functions
        - See also
    - Common table expressions (CTE)
        - How to do it...
            - Non-recursive CTE
            - Recursive CTE
    - Generated columns
        - How to do it...
    - Window functions
        - How to do it...
            - Row number
            - Partition results
            - Named windows
            - First, last, and nth values

4. Configuring MySQL
    - Introduction
    - Using config file
        - How to do it...
    - Using global and session variables
        - How to do it...
    - Using parameters with startup script
        - How to do it...
    - Configuring the parameters
        - How to do it...
            - data directory
            - innodb_buffer_pool_size
            - innodb_buffer_pool_instances
            - innodb_log_file_size
    - Changing the data directory
        - How to do it...

5. Transactions
    - Introduction
    - Performing transactions
        - How to do it...
            - Autocommit
    - Using savepoints
        - How to do it...
    - Isolation levels
        - How to do it...
            - Read uncommitted
            - Read committed
            - Repeatable read
            - Serializable
    - Locking
        - How to do it...
            - Locking queue

6. Binary Logging
    - Introduction
    - Using binary logging
        - How to do it...
            - Enabling binary logs
            - Disabling binary logs for a session
            - Move to the next log
            - Expire binary logs
    - Binary log format
        - How to do it...
        - See also
    - Extracting statements from a binary log
        - Getting ready
        - How to do it...
            - Observations
            - Extracting based on time and position
            - Extracting based on the database

            - Extracting a row event display
            - Rewriting a database name
            - Disabling a binary log for recovery
            - Displaying events in a binary log file
    - Ignoring databases to write to a binary log
        - How to do it...
            - Example 1
            - Example 2
    - Relocating binary logs
        - How to do it...

7. Backups
    - Introduction
    - Taking backups using mysqldump
        - How to do it...
            - Full backup of all databases
            - Point-in-time recovery
            - Dumping master binary coordinates
            - Specific databases and tables
            - Ignore tables
            - Specific rows
            - Backup from a remote server
            - Backup to rebuild another server with a different schem
              a
            - Only schema and no data
            - Only data and no schema
            - Backup for merging data with other server
                - REPLACE with new data
                - IGNORE data
    - Taking backups using mysqlpump
        - How to do it...
            - Parallel processing
            - Exclude/include database objects using regex

            - Backup users
            - Compressed backups
            - Faster reload
    - Taking backups using mydumper
        - How to do it...
            - Installation
            - Full backup
            - Consistent backup
            - Backup of a single table
            - Backup of specific databases using regex
            - Taking backup of a big table using mydumper
            - Non-blocking backup
            - Compressed backups
            - Backing up only data
    - Taking backups using flat files
        - How to do it...
    - Taking backups using XtraBackup
        - How to do it...
            - Installation
                - On CentOS/Red Hat/Fedora
                - On Debian/Ubuntu
    - Locking instances for backup
        - How to do it...
    - Binary log backup
        - How to do it...

8. Restoring Data
    - Introduction
    - Recovering from mysqldump and mysqlpump
        - How to do it...
        - There's more...
    - Recovering from mydumper using myloader
        - How to do it...
            - Recovering full database

            - Recover a single database
            - Recovering a single table
    - Recovering from flat file backup
        - How to do it...
    - Performing point-in-time recovery
        - How to do it...
            - mysqldump or mysqlpump
            - mydumper

9. Replication
    - Introduction
    - Setting up replication
        - How to do it...
    - Setting up master-master replication
        - How to do it...
    - Setting up multi-source replication
        - How to do it...
    - Setting up replication filters
        - How to do it...
            - Replicate a database only
            - Replicate specific tables
            - Ignore a database
            - Ignore specific tables
        - See also
    - Switching slave from master-slave to chain replication
        - How to do it...
    - Switching the slave from chain replication to master-slave
        - How to do it...
    - Setting up delayed replication
        - How to do it...
    - Setting up GTID replication
        - How to do it...
    - Setting up semi-synchronous replication
        - How to do it...

10. Table Maintenance
- Introduction
- Installing Percona Toolkit
  - How to do it...
    - On Debian/Ubuntu
    - On CentOS/Red Hat/Fedora
- Altering tables
  - How to do it...
- Moving tables across databases
  - How to do it...
- Altering tables using an online schema change tool
  - How it works...
  - How to do it...
- Archiving tables
  - How to do it...
    - Purging data
    - Archiving data
    - Copying data
  - See also
- Cloning tables
  - How to do it...
- Partitioning tables
  - How to do it...
    - RANGE partitioning
    - Removing partitioning
    - RANGE COLUMNS partitioning
    - LIST and LIST COLUMNS partitioning
    - HASH and LINEAR HASH partitioning
    - KEY and LINEAR KEY partitioning
    - Subpartitioning
- Partition pruning and selection
  - How to do it...
    - Partition pruning

Partition selection
Partition management
How to do it...
ADD partitions
Reorganizing partitions
DROP partitions
TRUNCATE partitions
Managing HASH and KEY partitions
Other operations
Partition information
How to do it...
Using SHOW CREATE TABLE
Using SHOW TABLE STATUS
Using EXPLAIN
Querying the INFORMATION_SCHEMA.PARTITION
S table
Efficiently managing time to live and soft delete rows
How it works...
How to do it...
11. Managing Tablespace
Introduction
Changing the number or size of InnoDB redo log files
How to do it...
Resizing the InnoDB system tablespace
How to do it...
Increasing the InnoDB system tablespace
Shrinking the InnoDB system tablespace
Creating file-per-table tablespaces outside the data directory
How to do it...
Copying file-per-table tablespaces to another instance
How to do it...
Copy full table
Copying individual partitions of a table

        - See also
    - Managing UNDO tablespace
        - How to do it...
    - Managing general tablespace
        - How to do it...
            - Create a general tablespace
            - Adding tables to a general tablespace
            - Moving non-partitioned tables between tablespaces
            - Managing partitioned tables in a general tablespace
            - Dropping general tablespace
    - Compressing InnoDB tables
        - How to do it...
            - Enabling Compression for file_per_table Tables
            - Disabling Compression for file_per_table Tables
            - Enabling Compression for General Tablespace

12. Managing Logs
    - Introduction
    - Managing the error log
        - How to do it...
            - Configuring the error log
            - Rotating the error log
            - Using the system log for logging
            - Error logging in JSON format
    - Managing the general query log and slow query log
        - How to do it...
            - General query log
            - Slow query log
            - Selecting query log output destinations
    - Managing the binary logs
        - How to do it...
13. Performance Tuning
    - Introduction
    - The explain plan

- How to do it...
  - Using EXPLAIN
    - Using EXPLAIN JSON
    - Using EXPLAIN for connection

Benchmarking queries and the server

- How to do it...

Adding indexes

- Primary key (clustered index) and secondary indexes
- How to do it...
  - Add index
  - UNIQUE index
  - Prefix index
  - Drop index
  - Index on generated columns

Invisible index

- How to do it...

Descending index

- How to do it...

Analyzing slow queries using pt-query-digest

- How to do it...
  - Slow query log
  - General query log
  - Process list
  - Binary log
  - TCP dump
- See also

Optimizing datatypes

- How to do it...

Removing duplicate and redundant indexes

- How to do it...
  - pt-duplicate-key-checker
  - mysqlindexcheck

Checking index usage

How to do it...
Controlling the query optimizer
How to do it...
optimizer_search_depth
How to know that the query is spending time in ev aluating plans?
optimizer_switch
Optimizer hints
Adjusting the optimizer cost model
Using index hints
How to do it...
Indexing for JSON using generated columns
How to do it...
Using resource groups
How to do it...
Alter and drop resource group
Using performance_schema
How to do it...
Enable/disable performance_schema
Enable/disable consumers and instruments
performance_schema tables
Using the sys schema
How to do it...
Statement by type (INSERT and SELECT) from each h ost
Statement by type from each user
Redundant indexes
Unused indexes
Statements executed from each host
Table statistics
Table statistics with buffer
Statement analysis
14. Security

Introduction
Securing installation
  How to do it...
    The FILE privilege
Restricting networks and users
  How to do it...
Password-less authentication using mysql_config_editor
  How to do it...
Resetting the root password
  How to do it...
    Using init-file
    Using --skip-grant-tables
Setting up encrypted connections using X509
  How to do it...
Setting up SSL replication
  How to do it...

Preface

MySQL is one of the most popular and widely used relational databases in the world today. With the recently released MySQL 8, it promises to be better and more efficient than ever, giving you high-performance querying results and easy configuration as an administrator.

Who this book is for

This book is for a wide range of readers. MySQL database administrators and developers who have worked on earlier versions of MySQL will learn about the features of MySQL 8 and how they can leverage them. For readers who worked on other RDBMSes such as Oracle, MSSQL, PostgreSQL, and Db2 this book will be a quick start guide to MySQL 8. For beginners, this book serves as a handbook; they can refer to the recipes and find quick solutions to their problems.

Most importantly, this book makes you *production-ready*. After reading this book, you will be confident in handling busy database servers with large datasets.

In my 10 years of experience with MySQL, I have witnessed small mistakes leading to major outages. In this book, many scenarios where a mistake can be made are covered and put under a warning label.

The topics are introduced in such a way that a beginner need not go back and forth to understand the concepts. A reference link to the MySQL documentation or any other source is provided for each topic, and the reader can refer to the link for more details.

Since this book is written to suit beginners as well, there may be a few recipes on topics you already know; feel free to skip them.

What this book covers

Practice makes a man perfect. But to practice, you need some knowledge and training. This book helps you with that. Most day-to-day and practical scenarios are covered in this book.

Chapter 1, *MySQL 8 - Installing and Upgrading*, describes how to install MySQL 8 on different flavors of Linux, upgrade to MySQL 8 from previous stable releases, and also downgrade from MySQL 8.

Chapter 2, *Using MySQL*, takes you through the basic uses of MySQL, such as creating databases and tables; inserting, updating, deleting, and selecting data in various ways; saving to different destinations; sorting and grouping results; joining tables; managing users; other database elements such as triggers, stored procedures, functions, and events; and getting metadata information.

Chapter 3, *Using MySQL (Advanced)*, covers the latest additions to MySQL 8, such as the JSON datatype, common table expressions, and window functions.

Chapter 4, *Configuring MySQL*, shows you how to configure MySQL and basic configuration parameters.

Chapter 5, *Transactions*, explains the four isolation levels of RDBMS and how to use MySQL for transactions.

Chapter 6, *Binary Logging*, demonstrates how to enable binary logging, various formats of binary logs, and how to retrieve data from binary logs.

Chapter 7, *Backups*, covers various types of backups, the pros and cons of each method, and which one to choose based on your requirements.

Chapter 8, *Restoring Data*, covers how to recover data from varies backups.

Chapter 9, *Replication*, explains how to set up various replication topologies. The recipes on switching a slave from master-slave to chain replication and switching a slave from chain replication to master-slave is something that will interest the readers.

Chapter 10, *Table Maintenance*, covers cloning tables. Managing big tables is something that this chapter will make you a maestro of. Installation and usage of third-party tools is also covered in this chapter.

Chapter 11, *Managing Tablespace*, deals with recipes that will teach the readers how to resize, create, copy, and manage tablespaces.

Chapter 12, *Managing Logs*, takes readers through error, general query, slow query, and binary logs.

Chapter 13, *Performance Tuning*, explains query and schema tuning in detail. There are ample recipes in the chapter that will cover this.

Chapter 14, *Security*, focuses on the aspects of security. Recipes on securing installation, restricting networks and users, setting and resetting of passwords, and much more in covered are detail.

To get the most out of this book

Basic knowledge of any Linux system makes it easy for you to understand this book.

Conventions used

There are a number of text conventions used throughout this book.

`CodeInText`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "MySQL has a dependency on the `libaio` library."

When we wish to draw your attention to a particular part of a command line statement, the relevant lines or items are set in bold:

```
shell> sudo yum repolist all | grep mysql8
mysql80-community/x86_64             MySQL 8.0 Community
Server  enabled:     16
mysql80-community-source             MySQL 8.0 Community
Server  disabled
```

Any command-line input or output is written as follows:

```
mysql> ALTER TABLE table_name REMOVE PARTITIONING;
```

Bold: Indicates a new term, an important word, or words that you see onscreen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "Select the Development Releases tab for getting MySQL 8.0 and the choose the OS and version."

Warnings or important notes appear like this.

Tips and tricks appear like this.

Sections

In this book, you will find several headings that appear frequently (*Getting ready*, *How to do it...*, *How it works...*, *There's more...*, and *See also*).

To give clear instructions on how to complete a recipe, use these sections as follows:

Getting ready

This section tells you what to expect in the recipe and describes how to set up any software or any preliminary settings required for the recipe.

How to do it…

This section contains the steps required to follow the recipe.

How it works…

This section usually consists of a detailed explanation of what happened in the previous section.

There's more…

This section consists of additional information about the recipe in order to make you more knowledgeable about the recipe.

See also

This section provides helpful links to other useful information for the recipe.

Get in touch

Feedback from our readers is always welcome.

General feedback: Email `feedback@packtpub.com` and mention the book title in the subject of your message. If you have questions about any aspect of this book, please email us at `questions@packtpub.com`.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit www.packtpub.com/submit-errata, selecting your book, clicking on the Errata Submission Form link, and entering the details.

Piracy: If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packtpub.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit authors.packtpub.com.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at Packt can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about Packt, please visit packtpub.com.

MySQL 8 - Installing and Upgrading

In this chapter, we will cover the following recipes:

- Installing MySQL using YUM/APT
- Installing MySQL 8.0 using RPM or DEB files
- Installing MySQL on Linux using Generic Binaries
- Starting or Stopping MySQL 8
- Uninstalling MySQL 8
- Managing MySQL Server with systemd
- Downgrading from MySQL 8.0
- Upgrading to MySQL 8.0
- Installing MySQL utilities

Introduction

In this chapter, you will learn about the installing, upgrading, and downgrading steps of MySQL 8. There are five different ways to install or upgrade; the three most widely-used installation methods are covered in this chapter:

- Software repositories (YUM or APT)
- RPM or DEB files
- Generic Binaries
- Docker (not covered)
- Source code compilation (not covered)

If you have already installed MySQL and want to upgrade, go through the upgrade steps in the *Upgrade to MySQL 8* section. If your installation is corrupt, go through the uninstallation steps also in the *Upgrade to MySQL 8* section.

Before installation, make a note of OS and CPU architecture. The convention followed is as follows:

MySQL Linux RPM package distribution identifiers

Distribution value	**Intended use**
el6, el7	Red Hat Enterprise Linux, Oracle Linux, CentOS 6 or 7
fc23, fc24, fc25	Fedora 23, 24, or 25

Distribution value	Intended use
sles12	SUSE Linux Enterprise Server 12

MySQL Linux RPM package CPU identifiers

CPU value	Intended processor type or family
i386, i586, i686	Pentium processor or better, 32-bit
x86_64	64-bit x86 processor
ia64	Itanium (IA-64) processor

MySQL Debian and Ubuntu 7 and 8 installation packages CPU identifiers

CPU value	Intended processor type or family
i386	Pentium processor or better, 32-bit
amd64	64-bit x86 processor

MySQL Debian 6 Installation package CPU identifiers

CPU value	**Intended processor type or family**
i686	Pentium processor or better, 32-bit
x86_64	64-bit x86 processor

Installing MySQL using YUM/APT

The most common and easiest way of installation is through software repositories where you add official Oracle MySQL repositories to your list and install MySQL through package management software.

There are mainly two types of repository software:

- YUM (Centos, Red Hat, Fedora and Oracle Linux)
- APT (Debian, Ubuntu)

How to do it...

Let's look at steps for installing MySQL 8 in the following ways:

Using YUM repositories

1. Find the Red Hat or CentOS version:

```
shell> cat /etc/redhat-release
CentOS Linux release 7.3.1611 (Core)
```

2. Add the MySQL Yum repository to your system's repository list. This is a one-time operation that can be performed by installing an RPM provided by MySQL.
   You can download the MySQL YUM Repository from http://dev.mysql.com/downloads/repo/yum/ and choose the file depending on your OS.
   Install the downloaded release package with the following command, replacing the name with the platform- and version-specific package name of the downloaded RPM package:

```
shell> sudo yum localinstall -y mysql57-community-release-el7-11.noarch.rpm
Loaded plugins: fastestmirror
Examining mysql57-community-release-el7-11.noarch.rpm: mysql57-community-
release-el7-11.noarch
Marking mysql57-community-release-el7-11.noarch.rpm to be installed
Resolving Dependencies
--> Running transaction check
---> Package mysql57-community-release.noarch 0:el7-11 will be installed
--> Finished Dependency Resolution
~
  Verifying  : mysql57-community-release-el7-11.noarch 1/1

Installed:
  mysql57-community-release.noarch 0:el7-11
Complete!
```

3. Or you can copy the link location and install directly using RPM (you can skip the next step after installing):

```
shell> sudo rpm -Uvh "https://dev.mysql.com/get/mysql57-community-release-
el7-11.noarch.rpm"
Retrieving https://dev.mysql.com/get/mysql57-community-release-el7-
11.noarch.rpm
Preparing...                          #################################
[100%]
Updating / installing...
   1:mysql57-community-release-el7-11 #################################
[100%]
```

4. Verify the installation:

```
shell> yum repolist enabled | grep 'mysql.*-community.*'
mysql-connectors-community/x86_64 MySQL Connectors Community
42
mysql-tools-community/x86_64      MySQL Tools Community
53
```

```
mysql57-community/x86_64          MySQL 5.7 Community Server
227
```

5. Set the release series. At the time of writing this book, MySQL 8 is not a **general availability** (**GA**) release. So MySQL 5.7 will be selected as the default release series. To install MySQL 8, you have to set the release series to 8:

```
shell> sudo yum repolist all | grep mysql
mysql-cluster-7.5-community/x86_64   MySQL Cluster 7.5 Community disabled
mysql-cluster-7.5-community-source   MySQL Cluster 7.5 Community disabled
mysql-cluster-7.6-community/x86_64   MySQL Cluster 7.6 Community disabled
mysql-cluster-7.6-community-source   MySQL Cluster 7.6 Community disabled
mysql-connectors-community/x86_64    MySQL Connectors Community  enabled:
42
mysql-connectors-community-source    MySQL Connectors Community  disabled
mysql-tools-community/x86_64         MySQL Tools Community       enabled:
53
mysql-tools-community-source         MySQL Tools Community - Sou disabled
mysql-tools-preview/x86_64           MySQL Tools Preview         disabled
mysql-tools-preview-source           MySQL Tools Preview - Sourc disabled
mysql55-community/x86_64             MySQL 5.5 Community Server  disabled
mysql55-community-source             MySQL 5.5 Community Server  disabled
mysql56-community/x86_64             MySQL 5.6 Community Server  disabled
mysql56-community-source             MySQL 5.6 Community Server  disabled
mysql57-community/x86_64             MySQL 5.7 Community Server  enabled:
227
mysql57-community-source             MySQL 5.7 Community Server  disabled
mysql80-community/x86_64             MySQL 8.0 Community Server  disabled
mysql80-community-source             MySQL 8.0 Community Server  disabled
```

6. Disable `mysql57-community` and enable `mysql80-community`:

```
shell> sudo yum install yum-utils.noarch -y
shell> sudo yum-config-manager --disable mysql57-community
shell> sudo yum-config-manager --enable mysql80-community
```

7. Verify that `mysql80-community` is enabled:

```
shell> sudo yum repolist all | grep mysql8
mysql80-community/x86_64             MySQL 8.0 Community Server  enabled:
16
mysql80-community-source             MySQL 8.0 Community Server  disabled
```

8. Install MySQL 8:

```
shell> sudo yum install -y mysql-community-server
Loaded plugins: fastestmirror
mysql-connectors-community | 2.5 kB  00:00:00
mysql-tools-community      | 2.5 kB  00:00:00
mysql80-community          | 2.5 kB  00:00:00
Loading mirror speeds from cached hostfile
 * base: mirror.web-ster.com
 * epel: mirrors.cat.pdx.edu
 * extras: mirrors.oit.uci.edu
 * updates: repos.lax.quadranet.com
Resolving Dependencies
```

```
~
Transaction test succeeded
Running transaction
  Installing : mysql-community-common-8.0.3-0.1.rc.el7.x86_64   1/4
  Installing : mysql-community-libs-8.0.3-0.1.rc.el7.x86_64     2/4
  Installing : mysql-community-client-8.0.3-0.1.rc.el7.x86_64   3/4
  Installing : mysql-community-server-8.0.3-0.1.rc.el7.x86_64   4/4
  Verifying  : mysql-community-libs-8.0.3-0.1.rc.el7.x86_64     1/4
  Verifying  : mysql-community-common-8.0.3-0.1.rc.el7.x86_64   2/4
  Verifying  : mysql-community-client-8.0.3-0.1.rc.el7.x86_64   3/4
  Verifying  : mysql-community-server-8.0.3-0.1.rc.el7.x86_64   4/4

Installed:
  mysql-community-server.x86_64 0:8.0.3-0.1.rc.el7
Dependency Installed:
  mysql-community-client.x86_64 0:8.0.3-0.1.rc.el7
  mysql-community-common.x86_64 0:8.0.3-0.1.rc.el7
  mysql-community-libs.x86_64 0:8.0.3-0.1.rc.el7

Complete!
```

9. You can check the installed packages using the following:

```
shell> rpm -qa | grep -i 'mysql.*8.*'
perl-DBD-MySQL-4.023-5.el7.x86_64
mysql-community-libs-8.0.3-0.1.rc.el7.x86_64
mysql-community-common-8.0.3-0.1.rc.el7.x86_64
mysql-community-client-8.0.3-0.1.rc.el7.x86_64
mysql-community-server-8.0.3-0.1.rc.el7.x86_64
```

Using APT repositories

1. Add the MySQL APT repository to your system's repository list. This is a one-time operation that can be performed by installing a `.deb` file provided by MySQL
   You can download the MySQL APT repository from http://dev.mysql.com/downloads/repo/apt/.
   Or you can copy the link location and use `wget` to download directly on to the server. You might need to install `wget` (`sudo apt-get install wget`):

   ```
   shell> wget "https://repo.mysql.com//mysql-apt-
   config_0.8.9-1_all.deb"
   ```

2. Install the downloaded release package with the following command, replacing the name with platform- and version-specific package name of the downloaded APT package:

   ```
   shell> sudo dpkg -i mysql-apt-config_0.8.9-1_all.deb
   (Reading database ... 131133 files and directories
   currently installed.)
   Preparing to unpack mysql-apt-config_0.8.9-1_all.deb
   ...
   Unpacking mysql-apt-config (0.8.9-1) over (0.8.9-1) ...
   Setting up mysql-apt-config (0.8.9-1) ...
   Warning: apt-key should not be used in scripts (called
   from postinst maintainerscript of the package mysql-
   apt-config)
   OK
   ```

3. During the installation of the package, you will be asked to choose the versions of the MySQL server and other components. Press *Enter* for selecting and the Up and Down keys for navigating.

Select MySQL Server and Cluster (Currently selected: mysql-5.7).
Select mysql-8.0 preview (At the time of writing, MySQL 8.0 is not GA). You might get a warning such as MySQL 8.0-RC Note that MySQL 8.0 is currently an RC. It should only be installed to preview upcoming features of MySQL, and is not recommended for use in production environments. (**RC** is short for **release candidate**).
If you want to change the release version, execute the following:

```
shell> sudo dpkg-reconfigure mysql-apt-config
```

4. Update package information from the MySQL APT repository with the following command (this step is mandatory):

```
shell> sudo apt-get update
```

5. Install MySQL. During installation, you'll need to provide a password for the root user for your MySQL installation. Remember the password; if you forget it, you'll have to reset the root password (refer to the *Resetting root password* section). This installs the package for the MySQL server, as well as the packages for the client and for the database common files:

```
shell> sudo apt-get install -y mysql-community-server
~
Processing triggers for ureadahead (0.100.0-19) ...
Setting up mysql-common (8.0.3-rc-1ubuntu14.04) ...
update-alternatives: using /etc/mysql/my.cnf.fallback
to provide /etc/mysql/my.cnf (my.cnf) in auto mode
Setting up mysql-community-client-core (8.0.3-rc-
1ubuntu14.04) ...
Setting up mysql-community-server-core (8.0.3-rc-
1ubuntu14.04) ...
~
```

6. Verify packages. `ii` indicates that the package is installed:

```
shell> dpkg -l | grep -i mysql
ii  mysql-apt-config            0.8.9-1
all   Auto configuration for MySQL APT Repo.
ii  mysql-client                8.0.3-rc-1ubuntu14.04
amd64 MySQL Client meta package depending on latest
version
ii  mysql-common                8.0.3-rc-1ubuntu14.04
amd64 MySQL Common
ii  mysql-community-client      8.0.3-rc-1ubuntu14.04
amd64 MySQL Client
ii  mysql-community-client-core 8.0.3-rc-1ubuntu14.04
amd64 MySQL Client Core Binaries
ii  mysql-community-server      8.0.3-rc-1ubuntu14.04
amd64 MySQL Server
ii  mysql-community-server-core 8.0.3-rc-1ubuntu14.04
amd64 MySQL Server Core Binaires
```

Installing MySQL 8.0 using RPM or DEB files

Installing MySQL using repositories requires access to public internet. As a security measure, most of the production machines are not connected to the internet. In that case, you can download the RPM or DEB files on the system administration and copy them to the production machine.

There are mainly two types of installation files:

- RPM (CentOS, Red Hat, Fedora, and Oracle Linux)
- DEB (Debian, Ubuntu)

There are multiple packages that you need to install. Here is a list and short description of each one:

- `mysql-community-server`: Database server and related tools.
- `mysql-community-client`: MySQL client applications and tools.
- `mysql-community-common`: Common files for server and client libraries.
- `mysql-community-devel`: Development header files and libraries for MySQL database client applications, such as the Perl MySQL module.
- `mysql-community-libs`: The shared libraries (`libmysqlclient.so*`) that certain languages and applications need to dynamically load and use MySQL.
- `mysql-community-libs-compat`: The shared libraries for older releases. Install this package if you have applications installed that are dynamically linked against older versions of MySQL

but you want to upgrade to the current version without breaking the library dependencies.

How to do it...

Let's look at how to do it using the following types of bundles:

Using the RPM bundle

1. Download the MySQL RPM tar bundle from the MySQL Downloads page, http://dev.mysql.com/downloads/mysql/, choosing your OS and CPU architecture. At the time of writing, MySQL 8.0 is not GA. If it is still in the development series, select the Development Releases tab for getting MySQL 8.0 and the choose the OS and version:

```
shell> wget 'https://dev.mysql.com/get/Downloads/MySQL-
8.0/mysql-8.0.3-0.1.rc.el7.x86_64.rpm-bundle.tar'
~
Saving to: ‘mysql-8.0.3-0.1.rc.el7.x86_64.rpm-
bundle.tar’
~
```

2. Untar the package:

```
shell> tar xfv mysql-8.0.3-0.1.rc.el7.x86_64.rpm-
bundle.tar
```

3. Install MySQL:

```
shell> sudo rpm -i mysql-community-{server-
8,client,common,libs}*
```

4. RPM cannot solve the dependency issues and the installation process might run issues. If you are facing such issues, use the `yum` command listed here (you should have access to dependent packages):

```
shell> sudo yum install mysql-community-{server-
8,client,common,libs}* -y
```

5. Verify the installation:

```
shell> rpm -qa | grep -i mysql-community
mysql-community-common-8.0.3-0.1.rc.el7.x86_64
mysql-community-libs-compat-8.0.3-0.1.rc.el7.x86_64
mysql-community-libs-8.0.3-0.1.rc.el7.x86_64
mysql-community-server-8.0.3-0.1.rc.el7.x86_64
mysql-community-client-8.0.3-0.1.rc.el7.x86_64
```

Using the APT bundle

1. Download the MySQL APT TAR from the MySQL Downloads page, http://dev.mysql.com/downloads/mysql/:

```
shell> wget "https://dev.mysql.com/get/Downloads/MySQL-
8.0/mysql-server_8.0.3-rc-1ubuntu16.04_amd64.deb-
bundle.tar"
~
Saving to: ‘mysql-server_8.0.3-rc-
1ubuntu16.04_amd64.deb-bundle.tar’
~
```

2. Untar the packages:

```
shell> tar -xvf mysql-server_8.0.3-rc-
1ubuntu16.04_amd64.deb-bundle.tar
```

3. Install the dependencies. You may need to install the `libaio1` package if it is not already installed:

```
shell> sudo apt-get install -y libaio1
```

4. Upgrade `libstdc++6` to the latest:

```
shell> sudo add-apt-repository ppa:ubuntu-toolchain-
r/test
shell> sudo apt-get update
shell> sudo apt-get upgrade -y libstdc++6
```

5. Upgrade `libmecab2` to the latest. If `universe` is not included, then add the following line to the end of the file (for example, `zesty`):

```
shell> sudo vi /etc/apt/sources.list
deb http://us.archive.ubuntu.com/ubuntu zesty main
```

```
universe

shell> sudo apt-get update
shell> sudo apt-get install libmecab2
```

6. Preconfigure the MySQL server package with the following command. It asks you to set the root password:

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

7. 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:

```
shell> sudo dpkg -i mysql-{common,community-client-
core,community-client,client,community-server-
core,community-server,server}_*.deb
```

8. Install the shared libraries:

```
shell> sudo dpkg -i libmysqlclient21_8.0.1-dmr-
1ubuntu16.10_amd64.deb
```

9. Verify the installation:

```
shell> dpkg -l | grep -i mysql
ii  mysql-client                8.0.3-rc-1ubuntu14.04
amd64 MySQL Client meta package depending on latest
version
ii  mysql-common                8.0.3-rc-1ubuntu14.04
amd64 MySQL Common
ii  mysql-community-client      8.0.3-rc-1ubuntu14.04
amd64 MySQL Client
ii  mysql-community-client-core 8.0.3-rc-1ubuntu14.04
amd64 MySQL Client Core Binaries
ii  mysql-community-server      8.0.3-rc-1ubuntu14.04
amd64 MySQL Server
ii  mysql-community-server-core 8.0.3-rc-1ubuntu14.04
```

```
amd64 MySQL Server Core Binaires
ii  mysql-server                8.0.3-rc-1ubuntu16.04
amd64 MySQL Server meta package depending on latest
version
```

Installing MySQL on Linux using Generic Binaries

Installing using the software packages requires some dependencies to be installed first and can conflict with other packages. In that case, you can install MySQL using the generic binaries available on the downloads page. Binaries are precompiled using advanced compilers and are built with the best possible options for optimal performance.

How to do it...

MySQL has a dependency on the `libaio` library. The `data directory` initialization, and subsequent server startup steps, will fail if this library is not installed locally.

On YUM-based systems:

```
shell> sudo yum install -y libaio
```

On APT-based systems:

```
shell> sudo apt-get install -y libaio1
```

Download the TAR binary from the MySQL Downloads page, at https://dev.mysql.com/downloads/mysql/, then choose Linux - Generic as the OS and select the version. You can download directly onto your server directly using the `wget` command:

```
shell> cd /opt
shell> wget "https://dev.mysql.com/get/Downloads/MySQL-
8.0/mysql-8.0.3-rc-linux-glibc2.12-x86_64.tar.gz"
```

Install MySQL using the following steps:

1. Add the `mysql` group and the `mysql` user. All the files and directories should be under the `mysql` user:

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

2. This is the installation location (you can change it to another location):

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

3. Untar the binary file. Keep the untarred binary file at the same location and symlink it to the installation location. In this way, you can keep multiple versions and it is very easy to upgrade. For example, you can download another version and untar it to a different location; while upgrading, all you need to do is to change the symlink:

```
shell> sudo tar zxvf /opt/mysql-8.0.3-rc-linux-
glibc2.12-x86_64.tar.gz
mysql-8.0.3-rc-linux-glibc2.12-x86_64/bin/myisam_ftdump
mysql-8.0.3-rc-linux-glibc2.12-x86_64/bin/myisamchk
```

```
mysql-8.0.3-rc-linux-glibc2.12-x86_64/bin/myisamlog
mysql-8.0.3-rc-linux-glibc2.12-x86_64/bin/myisampack
mysql-8.0.3-rc-linux-glibc2.12-x86_64/bin/mysql
~
```

4. Make the symlink:

```
shell> sudo ln -s mysql-8.0.3-rc-linux-glibc2.12-x86_64
mysql
```

5. Create the necessary directories and change the ownership to `mysql`:

```
shell> cd mysql
shell> sudo mkdir mysql-files
shell> sudo chmod 750 mysql-files
shell> sudo chown -R mysql .
shell> sudo chgrp -R mysql .
```

6. Initialize `mysql`, which generates a temporary password:

```
shell> sudo bin/mysqld --initialize --user=mysql
~
2017-12-02T05:55:10.822139Z 5 [Note] A temporary
```

```
password is generated for root@localhost: Aw=ee.rf(6Ua
~
```

7. Set up the RSA for SSL. Refer to Chapter 14, *Setting up Encrypted Connections using X509 Section*, for more details on SSL. Note that a temporary password is generated for `root@localhost: eJQdj8C*qVMq`:

```
shell> sudo bin/mysql_ssl_rsa_setup
Generating a 2048 bit RSA private key
...........+++
....................................+++
writing new private key to 'ca-key.pem'
-----
Generating a 2048 bit RSA private key
.......................................................
....+++
...........................................+++
writing new private key to 'server-key.pem'
-----
Generating a 2048 bit RSA private key
.....+++
..........................+++
writing new private key to 'client-key.pem'
-----
```

8. Change the ownership of binaries to `root` and `data` files to `mysql`:

```
shell> sudo chown -R root .
shell> sudo chown -R mysql data mysql-files
```

9. Copy the startup script to `init.d`:

```
shell> sudo cp support-files/mysql.server
/etc/init.d/mysql
```

10. Export the binary of `mysql` to the `PATH` environment variable:

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

11. Refer to *Starting or Stopping MySQL 8* section to start MySQL.

After installation, you will get the following directories inside `/usr/local/mysql`:

Directory	**Contents of directory**
`bin`	`mysqld` server, client, and utility programs
`data`	Log files, databases
`docs`	MySQL manual in info format
`man`	Unix manual pages
`include`	Include (header) files
`lib`	Libraries
`share`	Miscellaneous support files, including error messages, sample configuration files, SQL for database installation

There's more...

There are other installation methods, such as:

1. Compiling from the source code. You can compile and build MySQL from the source code provided by Oracle where you have the flexibility to customize build parameters, compiler optimizations, and the installation location. It is highly recommended to use precompiled binaries provided by Oracle, unless you want specific compiler options or you are debugging MySQL.
   This method is not covered as it is used very rarely and it requires several development tools, which is beyond the scope of this book. For installation through source code, you can refer to the reference manual, at https://dev.mysql.com/doc/refman/8.0/en/source-installation.html.
2. Using Docker. The MySQL server can also be installed and managed using Docker image. Refer to https://hub.docker.com/r/mysql/mysql-server/ for installation, configuration, and also how to use MySQL under Docker.

Starting or Stopping MySQL 8

After the installation is completed, you can start/stop MySQL using the following commands, which vary from different platforms and installation methods. `mysqld` is the `mysql` server process. All the startup methods invoke the `mysqld` script.

How to do it...

Let's look at it in detail. Along with the starting and stopping, we will also learn something about checking the status of the server. Let's see how.

Starting the MySQL 8.0 server

You can start the server with the following commands:

1. Using `service`:

```
shell> sudo service mysql start
```

2. Using `init.d`:

```
shell> sudo /etc/init.d/mysql start
```

3. If you do not find the startup script (when you are doing binary installation), you can copy from the location where you untarred.

```
shell> sudo cp /usr/local/mysql/support-
files/mysql.server /etc/init.d/mysql
```

4. If your installation includes `systemd` support:

```
shell> sudo systemctl start mysqld
```

5. If the `systemd` support is not there, MySQL can be started using `mysqld_safe`. `mysqld_safe` is the launcher script for `mysqld` that safeguards the `mysqld` process. If `mysqld` is killed, `mysqld_safe` attempts to start the process again:

```
shell> sudo mysqld_safe --user=mysql &
```

After start,

1. The server is initialized.

2. The SSL certificate and key files are generated in the `data directory`.
3. The `validate_password` plugin is installed and enabled.
4. A superuser account, `root'@'localhost`, is created. A password for the superuser is set and stored in the error log file (not for binary installation). To reveal it, use the following command:

```
shell> sudo  grep "temporary password"
/var/log/mysqld.log
2017-12-02T07:23:20.915827Z 5 [Note] A temporary
password is generated for root@localhost: bkvotsG:h6jD
```

You can connect to MySQL using that temporary password.

```
shell> mysql -u root -pbkvotsG:h6jD
mysql: [Warning] Using a password on the command line
interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or
\g.
Your MySQL connection id is 7
Server version: 8.0.3-rc-log

Copyright (c) 2000, 2017, Oracle and/or its affiliates.
All rights reserved.

Oracle is a registered trademark of Oracle Corporation
and/or its
affiliates. Other names may be trademarks of their
respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the
current input statement.

mysql>
```

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

```
# You will be prompted for a password, enter the one
you got from the previous step

mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY
'NewPass4!';
Query OK, 0 rows affected (0.01 sec)

# password should contain at least one Upper case
letter, one lowercase letter, one digit, and one
special character, and that the total password length
is at least 8 characters
```

Stopping the MySQL 8.0 server

Stopping MySQL and checking the status are similar to starting it, except for the change of one word:

1. Using `service`:

```
shell> sudo service mysqld stop
Redirecting to /bin/systemctl stop  mysqld.service
```

2. Using `init.d`:

```
shell> sudo /etc/init.d/mysql stop
[ ok ] Stopping mysql (via systemctl): mysql.service.
```

3. If your installation includes the `systemd` support (refer to the *Managing MySQL Server with systemd* section):

```
shell> sudo systemctl stop mysqld
```

4. Using `mysqladmin`:

```
shell> mysqladmin -u root -p shutdown
```

Checking the status of the MySQL 8.0 server

1. Using `service`:

```
shell> sudo systemctl status mysqld
● mysqld.service - MySQL Server
   Loaded: loaded
(/usr/lib/systemd/system/mysqld.service; enabled;
vendor preset: disabled)
  Drop-In: /etc/systemd/system/mysqld.service.d
           └─override.conf
   Active: active (running) since Sat 2017-12-02
07:33:53 UTC; 14s ago
     Docs: man:mysqld(8)
           http://dev.mysql.com/doc/refman/en/using-
systemd.html
  Process: 10472 ExecStart=/usr/sbin/mysqld --daemonize
--pid-file=/var/run/mysqld/mysqld.pid $MYSQLD_OPTS
(code=exited, status=0/SUCCESS)
  Process: 10451
ExecStartPre=/usr/bin/mysqld_pre_systemd (code=exited,
status=0/SUCCESS)
 Main PID: 10477 (mysqld)
   CGroup: /system.slice/mysqld.service
           └─10477 /usr/sbin/mysqld --daemonize --pid-
file=/var/run/mysqld/mysqld.pid --general_log=1

Dec 02 07:33:51 centos7 systemd[1]: Starting MySQL
Server...
Dec 02 07:33:53 centos7 systemd[1]: Started MySQL
Server.
```

2. Using `init.d`:

```
shell> sudo /etc/init.d/mysql status
● mysql.service - LSB: start and stop MySQL
   Loaded: loaded (/etc/init.d/mysql; bad; vendor
preset: enabled)
```

```
   Active: inactive (dead)
     Docs: man:systemd-sysv-generator(8)

Dec 02 06:01:00 ubuntu systemd[1]: Starting LSB: start
and stop MySQL...
Dec 02 06:01:00 ubuntu mysql[20334]: Starting MySQL
Dec 02 06:01:00 ubuntu mysql[20334]:  *
Dec 02 06:01:00 ubuntu systemd[1]: Started LSB: start
and stop MySQL.
Dec 02 06:01:00 ubuntu mysql[20334]: 2017-12-
02T06:01:00.969284Z mysqld_safe A mysqld process
already exists
Dec 02 06:01:55 ubuntu systemd[1]: Stopping LSB: start
and stop MySQL...
Dec 02 06:01:55 ubuntu mysql[20445]: Shutting down
MySQL
Dec 02 06:01:57 ubuntu mysql[20445]: .. *
Dec 02 06:01:57 ubuntu systemd[1]: Stopped LSB: start
and stop MySQL.
Dec 02 07:26:33 ubuntu systemd[1]: Stopped LSB: start
and stop MySQL.
```

3. If your installation includes the `systemd` support (refer to the *Managing MySQL Server with systemd* section):

```
shell> sudo systemctl status mysqld
```

Uninstalling MySQL 8

If you have messed up with installation or you do not want MySQL 8 version, you can uninstall using the following steps. Before uninstalling, make sure to make backup files (refer to Chapter 7, *Backups*), if required, and stop MySQL.

How to do it...

Uninstalling will be dealt in a different way on different systems. Let's look at how.

On YUM-based systems

1. Check whether there are any existing packages:

```
shell> rpm -qa | grep -i mysql-community
mysql-community-libs-8.0.3-0.1.rc.el7.x86_64
mysql-community-common-8.0.3-0.1.rc.el7.x86_64
mysql-community-client-8.0.3-0.1.rc.el7.x86_64
mysql-community-libs-compat-8.0.3-0.1.rc.el7.x86_64
mysql-community-server-8.0.3-0.1.rc.el7.x86_64
```

2. Remove the packages. You may be notified that there are other packages dependent on MySQL. If you plan on installing MySQL again, you can ignore the warning by passing the `--nodeps` option:

```
shell> rpm -e <package-name>
```

   For example:

```
shell> sudo rpm -e mysql-community-server
```

3. To remove all packages:

```
shell> sudo rpm -qa | grep -i mysql-community | xargs
sudo rpm -e --nodeps
warning: /etc/my.cnf saved as /etc/my.cnf.rpmsave
```

On APT-based systems

1. Check whether there are any existing packages:

```
shell> dpkg -l | grep -i mysql
```

2. Remove the packages using the following:

```
shell> sudo apt-get remove mysql-community-server
mysql-client mysql-common mysql-community-client mysql-
community-client-core mysql-community-server mysql-
community-server-core -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages will be REMOVED:
  mysql-client mysql-common mysql-community-client
mysql-community-client-core mysql-community-server
mysql-community-server-core mysql-server
0 upgraded, 0 newly installed, 7 to remove and 341 not
upgraded.
After this operation, 357 MB disk space will be freed.
(Reading database ... 134358 files and directories
currently installed.)
Removing mysql-server (8.0.3-rc-1ubuntu16.04) ...
Removing mysql-community-server (8.0.3-rc-1ubuntu16.04)
...
update-alternatives: using /etc/mysql/my.cnf.fallback
to provide /etc/mysql/my.cnf (my.cnf) in auto mode
Removing mysql-client (8.0.3-rc-1ubuntu16.04) ...
Removing mysql-community-client (8.0.3-rc-1ubuntu16.04)
...
Removing mysql-common (8.0.3-rc-1ubuntu16.04) ...
Removing mysql-community-client-core (8.0.3-rc-
1ubuntu16.04) ...
Removing mysql-community-server-core (8.0.3-rc-
1ubuntu16.04) ...
Processing triggers for man-db (2.7.5-1) ...
```

   Or remove them using:

```
shell> sudo apt-get remove --purge mysql-\* -y
shell> sudo apt-get autoremove -y
```

3. Verify that the packages are uninstalled:

```
shell> dpkg -l | grep -i mysql
ii  mysql-apt-config        0.8.9-1              all
Auto configuration for MySQL APT Repo.
rc  mysql-common            8.0.3-rc-1ubuntu16.04 amd64
MySQL Common
rc  mysql-community-client  8.0.3-rc-1ubuntu16.04 amd64
MySQL Client
rc  mysql-community-server  8.0.3-rc-1ubuntu16.04 amd64
MySQL Server
```

`rc` indicates that the packages have been removed (`r`), and only config files (`c`) have been kept.

Uninstalling Binaries

It is very simple to uninstall a binary installation. All you need to do is to remove the symlink:

1. Change the directory to the installation path:

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

2. Check where `mysql` is pointing to, which will show the path it is referencing to:

```
shell> sudo ls -lh mysql
```

3. Remove `mysql`:

```
shell> sudo rm mysql
```

4. Remove the binaries (optional):

```
shell> sudo rm -f /opt/mysql-8.0.3-rc-linux-glibc2.12-
x86_64.tar.gz
```

Managing the MySQL Server with systemd

If you install MySQL using an RPM or Debian package server, startup and shutdown is managed by `systemd`. On platforms for which the `systemd` support for MySQL is installed, `mysqld_safe`, `mysqld_multi`, and `mysqld_multi.server` are not installed. MySQL server startup and shutdown is managed by `systemd` using the `systemctl` command. You need to configure `systemd` as follows.

RPM-based systems use the `mysqld.service` files, and APT-based systems use the `mysql.server` files.

How to do it...

1. Create a localized `systemd` configuration file:

```
shell> sudo mkdir -pv
/etc/systemd/system/mysqld.service.d
```

2. Create/open the `conf` file:

```
shell> sudo vi
/etc/systemd/system/mysqld.service.d/override.conf
```

3. Enter the following:

```
[Service]
LimitNOFILE=max_open_files (ex: 102400)
PIDFile=/path/to/pid/file (ex:
/var/lib/mysql/mysql.pid)
Nice=nice_level (ex: -10)
Environment="LD_PRELOAD=/path/to/malloc/library"
Environment="TZ=time_zone_setting"
```

4. Reload `systemd`:

```
shell> sudo systemctl daemon-reload
```

5. For temporary changes, you can reload without editing the `conf` file:

```
shell> sudo systemctl set-environment MYSQLD_OPTS="--
general_log=1"
or unset using
shell> sudo systemctl unset-environment MYSQLD_OPTS
```

6. After modifying the `systemd` environment, restart the server to make the changes effective.

Enable `mysql.serviceshell> sudo systemctl`, and enable `mysql.service`:

```
shell> sudo systemctl unmask mysql.service
```

7. Restart `mysql`:
   On RPM platforms:

```
shell> sudo systemctl restart mysqld
```

   On Debian platforms:

```
shell> sudo systemctl restart mysql
```

Downgrading from MySQL 8.0

If your application is not performing as expected, you can always downgrade to a previous GA release (MySQL 5.7). Before downgrading, it is recommended to take a logical backup (refer to Chapter 7, *Backups*). Note that you can downgrade by only one previous release. Suppose that you want to downgrade from MySQL 8.0 to MySQL 5.6, you have to downgrade to MySQL 5.7, and then from MySQL 5.7 to MySQL 5.6.

You can do it in two ways:

- In-place downgrade (downgrades within MySQL 8)
- Logical downgrade

How to do it...

In the following subsections, you will be learning how to handle the installation/uninstallation/upgrade/downgrade using various repositories, bundles, and so on.

In-place Downgrades

For downgrades between the GA status releases within MySQL 8.0 (note that you cannot downgrade to MySQL 5.7 using this method):

1. Shut down the old MySQL version
2. Replace the MySQL 8.0 binaries or older binaries
3. Restart MySQL on the existing `data directory`
4. Run the `mysql_upgrade` utility

Using YUM repositories

1. Prepare MySQL for a slow shutdown, which ensures that the undo logs are empty and data files are fully prepared in case of file format differences between releases:

```
mysql> SET GLOBAL innodb_fast_shutdown = 0;
```

2. Shut down the `mysql` server as described in the *Stopping MySQL 8.0 Server* section:

```
shell> sudo systemctl stop mysqld
```

3. Remove the `InnoDB` redo log files (the `ib_logfile*` files) from the `data directory` to avoid downgrade issues related to redo log file format changes that may have occurred between releases:

```
shell> sudo rm -rf /var/lib/mysql/ib_logfile*
```

4. Downgrade MySQL. To downgrade the server, you need to uninstall MySQL 8.0, as described in the *Uninstalling MySQL 8* section. The configuration files are automatically stored as backup.

   List the available versions:

```
shell> sudo yum list mysql-community-server
```

   Downgrades are tricky; it is better to remove the existing packages before downgrading:

```
shell> sudo rpm -qa | grep -i mysql-community | xargs
sudo rpm -e --nodeps
warning: /etc/my.cnf saved as /etc/my.cnf.rpmsave
```

Install the older version:

```
shell> sudo yum install -y mysql-community-server-
<version>
```

Using APT Repositories

1. Reconfigure MySQL and choose the older version:

```
shell> sudo dpkg-reconfigure mysql-apt-config
```

2. Run `apt-get update`:

```
shell> sudo apt-get update
```

3. Remove the current version:

```
shell> sudo apt-get remove mysql-community-server
mysql-client mysql-common mysql-community-client mysql-
community-client-core mysql-community-server mysql-
community-server-core -y

shell> sudo apt-get autoremove
```

4. Install the older version (autoselected since you have reconfigured):

```
shell> sudo apt-get install -y mysql-server
```

Using the RPM or APT bundle

Uninstall the existing packages (refer to the *Uninstalling MySQL 8* section) and install the new packages, which can be downloaded from the MySQL Downloads (refer to the *Installing MySQL 8.0 using RPMs or DEB files* section).

Using Generic Binaries

If you have installed MySQL through binaries, you have to remove the symlink to the old version (refer to the *Uninstalling MySQL 8* section) and do a fresh installation (refer to the *Installing MySQL on Linux Using Generic Binaries* section):

1. Start the server as described in the *Starting or Stopping MySQL 8* section. Please note that the start procedure is the same for all the versions.
2. Run the `mysql_upgrade` utility:

```
shell> sudo mysql_upgrade -u root -p
```

3. Restart the MySQL server to ensure that any changes made to the system tables take effect:

```
shell> sudo systemctl restart mysqld
```

Logical Downgrades

Here is an outline of the steps:

1. Export existing data from the MySQL 8.0 version using logical backup (refer to Chapter 7, *Backups* for logical backup methods)
2. Install MySQL 5.7
3. Load the dump file into the MySQL 5.7 version (refer to Chapter 8, *Restoring Data* for restoring methods)
4. Run the `mysql_upgrade` utility

Here are the detailed steps:

1. You need to take logical backup of the database. (refer to Chapter 7, *Backups* for a quicker backup called `mydumper`):

```
shell> mysqldump -u root -p --add-drop-table --routines
--events --all-databases --force > mysql80.sql
```

2. Shut down the MySQL server as described in the *Starting or Stopping MySQL 8* section.
3. Move the `data directory`. Instead of restoring the SQL backup (in step 1), we can move back the `data directory` if you want to keep MySQL 8:

```
shell> sudo mv /var/lib/mysql /var/lib/mysql80
```

4. Downgrade MySQL. To downgrade the server, we need to uninstall MySQL 8. The configuration files are automatically backed up.

Using YUM Repositories

After the uninstallation, install the older version:

1. Switch the repositories:

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

2. Verify that `mysql57-community` is enabled:

```
shell> yum repolist enabled | grep "mysql.*-
community.*"
!mysql-connectors-community/x86_64 MySQL Connectors
Community                 42
!mysql-tools-community/x86_64      MySQL Tools
Community                      53
!mysql57-community/x86_64          MySQL 5.7 Community
Server                227
```

3. Downgrades are tricky; it is better to remove the existing packages before downgrading:

```
shell> sudo rpm -qa | grep -i mysql-community | xargs
sudo rpm -e --nodeps
warning: /etc/my.cnf saved as /etc/my.cnf.rpmsave
```

4. List the available versions:

```
shell> sudo yum list mysql-community-server
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: mirror.rackspace.com
 * epel: mirrors.develooper.com
 * extras: centos.s.uw.edu
 * updates: mirrors.syringanetworks.net
```

```
Available Packages
mysql-community-server.x86_64   5.7.20-1.el7
mysql57-community
```

5. Install MySQL 5.7:

```
shell> sudo yum install -y mysql-community-server
```

Using APT Repositories

1. Reconfigure `apt` to switch to MySQL 5.7:

```
shell> sudo dpkg-reconfigure mysql-apt-config
```

2. Run `apt-get update`:

```
shell> sudo apt-get update
```

3. Remove the current version:

```
shell> sudo apt-get remove mysql-community-server
mysql-client mysql-common mysql-community-client mysql-
community-client-core mysql-community-server mysql-
community-server-core -y
shell> sudo apt-get autoremove
```

4. Install MySQL 5.7:

```
shell> sudo apt-get install -y mysql-server
```

Using RPM or APT bundles

Uninstall the existing packages (refer to the *Uninstalling MySQL 8* section) and install the new packages, which can be downloaded from MySQL Downloads (refer to the *Installing MySQL 8 using RPM or DEB files* section).

Using Generic Binaries

If you have installed MySQL through binaries, you have to remove the symlink to the old version (refer to the *Uninstalling MySQL 8* section) and do a fresh installation (refer to the *Installing MySQL on Linux using Generic Binaries* section).

Once you have downgraded MySQL, you have to restore the backup and run the `mysql_upgrade` utility:

1. Start MySQL (refer to the *Starting or Stopping MySQL 8* section). You need to reset the password again.
2. Restore the backup (this may take a long time, depending up on the size of backup). Refer to Chapter 8, *Restoring Data*, for a quick restoration method called `myloader`:

```
shell> mysql -u root -p < mysql80.sql
```

3. Run `mysql_upgrade`:

```
shell> mysql_upgrade -u root -p
```

4. Restart the MySQL server to ensure that any changes made to the system tables take effect. Refer to the *Starting or Stopping MySQL 8* section:

```
shell> sudo /etc/init.d/mysql restart
```

Upgrading to MySQL 8.0

MySQL 8 uses a global `data dictionary` containing information about database objects in transactional tables. In previous versions, the dictionary data was stored in metadata files and non-transactional system tables. You need to upgrade your `data directory` from the file-based structure to the data-dictionary structure.

Just like a downgrade, you can upgrade using two methods:

- In-place upgrade
- Logical upgrade

You also should check a few prerequisites before the upgrade.

Getting ready

1. Check for obsolete datatypes or triggers that have a missing or empty definer or an invalid creation context:

```
shell> sudo mysqlcheck -u root -p --all-databases --
check-upgrade
```

2. There must be no partitioned tables that use a storage engine that does not have native partitioning support. To identify these tables, execute this query:

```
shell> SELECT TABLE_SCHEMA, TABLE_NAME FROM
INFORMATION_SCHEMA.TABLES WHERE ENGINE NOT IN
('innodb', 'ndbcluster') AND CREATE_OPTIONS LIKE
'%partitioned%';
```

   If there are any of these tables, change them to `InnoDB`:

```
mysql> ALTER TABLE table_name ENGINE = INNODB;
```

   Or remove the partitioning:

```
mysql> ALTER TABLE table_name REMOVE PARTITIONING;
```

3. 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:

```
mysql> SELECT TABLE_SCHEMA, TABLE_NAME FROM
INFORMATION_SCHEMA.TABLES WHERE LOWER(TABLE_SCHEMA) =
'mysql' and LOWER(TABLE_NAME) IN ('catalogs',
'character_sets', 'collations', 'column_type_elements',
'columns', 'events', 'foreign_key_column_usage',
'foreign_keys', 'index_column_usage',
'index_partitions', 'index_stats', 'indexes',
'parameter_type_elements', 'parameters', 'routines',
'schemata', 'st_spatial_reference_systems',
'table_partition_values', 'table_partitions',
'table_stats', 'tables', 'tablespace_files',
'tablespaces', 'triggers', 'version',
'view_routine_usage', 'view_table_usage');
```

4. There must be no tables that have foreign key constraint names longer than 64 characters. To identify tables with constraint names that are too long, execute this query:

```
mysql> SELECT CONSTRAINT_SCHEMA, TABLE_NAME,
CONSTRAINT_NAME FROM
INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS WHERE
LENGTH(CONSTRAINT_NAME) > 64;
```

5. Tables not supported by MySQL 8.0, such as `ndb`, should be moved to `InnoDB`:

```
mysql> ALTER TABLE tablename ENGINE=InnoDB;
```

How to do it...

Just like the previous recipe, the following subsections will take you through the details with various systems, bundles, and so on.

In-place upgrades

Here is an outline of the steps:

1. Shut down the old MySQL version.
2. Replace the old MySQL binaries or packages with the new ones (detailed steps for different types of installation methods are covered).
3. Restart MySQL on the existing `data directory`.
4. Run the `mysql_upgrade` utility.
5. In the MySQL 5.7 server, if there are encrypted `InnoDB` tablespaces, rotate the `keyring` master key by executing this statement:

```
mysql> ALTER INSTANCE ROTATE INNODB MASTER KEY;
```

Here are the detailed steps:

1. Configure your MySQL 5.7 server to perform a slow shutdown. With a slow shutdown, `InnoDB` performs a full purge and change buffer merge before shutting down, which ensures that the undo logs are empty and the data files are fully prepared in case of file format differences between releases. This step is the most important because, without it, you will end up with the following error:

```
[ERROR] InnoDB: Upgrade after a crash is not supported.
```

   This redo log was created with MySQL 5.7.18. Please follow the instructions at http://dev.mysql.com/doc/refman/8.0/en/upgrading.html:

```
mysql> SET GLOBAL innodb_fast_shutdown = 0;
```

2. Shut down the MySQL server as described in the *Starting or Stopping MySQL 8* section.

Upgrade the MySQL binaries or packages.

YUM-based systems

1. Switch the repositories:

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

2. Verify that `mysql80-community` is enabled:

```
shell> sudo yum repolist all | grep mysql8
mysql80-community/x86_64             MySQL 8.0
Community Server  enabled:     16
mysql80-community-source             MySQL 8.0
Community Server  disabled
```

3. Run the yum update:

```
shell> sudo yum update mysql-server
```

APT-based systems

1. Reconfigure the `apt` to switch to MySQl 8.0:

```
shell> sudo dpkg-reconfigure mysql-apt-config
```

2. Run `apt-get update`:

```
shell> sudo apt-get update
```

3. Remove the current version:

```
shell> sudo apt-get remove mysql-community-server
mysql-client mysql-common mysql-community-client mysql-
community-client-core mysql-community-server mysql-
community-server-core -y
shell> sudo apt-get autoremove
```

4. Install MySQL 8:

```
shell> sudo apt-get update
shell> sudo apt-get install mysql-server
shell> sudo apt-get install libmysqlclient21
```

Using RPM or APT bundles

Uninstall the existing packages (refer to the *Uninstalling MySQL 8* section) and install the new packages, which can be downloaded from MySQL Downloads (refer to the *Installing MySQL 8.0 using RPM or DEB files* section).

Using Generic Binaries

If you have installed MySQL through binaries, you have to remove the symlink to the old version (refer to the *Uninstalling MySQL 8* section) and do a fresh installation (refer to the *Installing MySQL on Linux using generic binaries* section).

Start the MySQL 8.0 server (refer to the *Starting or Stopping MySQL 8 to start MySQL* section). If there are encrypted `InnoDB` tablespaces, use the `--early-plugin-load` option to load the `keyring` plugin.

The server 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, event scheduler events). The server also removes files that previously were used for metadata storage. For example, after upgrading, you will notice that your tables no longer have `.frm` files.

The server creates a directory named `backup_metadata_57` and moves the files used by MySQL 5.7 into it. The server renames the `event` and `proc` tables to `event_backup_57` and `proc_backup_57`. If this upgrade 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.

Run the `mysql_upgrade` utility:

```
shell> sudo mysql_upgrade -u root -p
```

`mysql_upgrade` examines all tables in all databases for incompatibilities with the current version of MySQL. It makes any remaining 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. `mysql_upgrade` also brings the performance schema, `INFORMATION_SCHEMA`, and `sys schema` objects up to date for MySQL 8.0.

Restart the MySQL server (refer to the *Starting or Stopping MySQL 8 to start MySQL* section).

Logical Upgrades

Here is an outline of the steps:

1. Export existing data from the old MySQL version using `mysqldump`
2. Install the new MySQL version
3. Load the dump file into the new MySQL version
4. Run the `mysql_upgrade` utility

Here are the detailed steps:

1. You need to take a logical backup of the database (refer to Chapter 7, *Backups* for a quicker backup called `mydumper`):

```
shell> mysqldump -u root -p --add-drop-table --routines
--events --all-databases --ignore-
table=mysql.innodb_table_stats --ignore-
table=mysql.innodb_index_stats --force > data-for-
upgrade.sql
```

2. Shut down the MySQL server (refer to the *Starting or Stopping MySQL 8* section).
3. Install the new MySQL version (refer to the methods mentioned in the *In-place upgrades* section).
4. Start the MySQL server (refer to the *Starting or Stopping MySQL 8* section).
5. Reset the temporary `root` password:

```
shell> mysql -u root -p
Enter password: **** (enter temporary root password
from error log)
```

```
mysql> ALTER USER USER() IDENTIFIED BY 'your new
password';
```

6. Restore the backup (this may take a long time depending up on the size of the backup). Refer to Chapter 8, *Restoring Data* for a quick restoration method called `myloader`:

```
shell> mysql -u root -p --force < data-for-upgrade.sql
```

7. Run the `mysql_upgrade` utility:

```
shell> sudo mysql_upgrade -u root -p
```

8. Restart the MySQL server (refer to the *Starting or Stopping MySQL 8* section).

Installing MySQL utilities

MySQL utilities gives you very handy tools to smoothly carry out day-to-day operations without much manual effort.

How to do it...

It can be installed on YUM-based and APT-based systems in the following manner. Let's take a look.

On YUM-based systems

Download the files from the MySQL downloads page, https://dev.mysql.com/downloads/utilities/, by selecting Red Hat Enterprise Linux/Oracle Linux, or directly from this link, using `wget`:

```
shell> wget
https://cdn.mysql.com//Downloads/MySQLGUITools/mysql-
utilities-1.6.5-1.el7.noarch.rpm

shell> sudo yum localinstall -y mysql-utilities-1.6.5-
1.el7.noarch.rpm
```

On APT-based systems

Download the files from the MySQL Downloads page, https://dev.mysql.com/downloads/utilities/, by selecting Ubuntu Linux, or directly from this link, using `wget`:

```
shell> wget
"https://cdn.mysql.com//Downloads/MySQLGUITools/mysql-
utilities_1.6.5-1ubuntu16.10_all.deb"
shell> sudo dpkg -i mysql-utilities_1.6.5-
1ubuntu16.10_all.deb
shell> sudo apt-get install -f
```

Using MySQL

In this chapter, we will cover the following recipes:

- Connecting to MySQL using the command-line client
- Creating databases
- Creating tables
- Inserting, updating, and deleting rows
- Loading sample data
- Selecting data
- Sorting results
- Grouping results (aggregate functions)
- Creating users
- Granting and revoking access to users
- Selecting data into a file and table
- Loading data into a table
- Joining tables
- Stored procedures
- Functions
- Triggers
- Views
- Events
- Getting information about databases and tables

Introduction

We are going to learn a lot of things in the following recipes. Let's take a look at each one in detail.

Connecting to MySQL using the command-line client

So far, you have learned how to install MySQL 8.0 on various platforms. Along with the installation, you will get the command-line client utility called `mysql`, which we use to connect to any MySQL server.

Getting ready

First you need to know to which server you need to connect. If you have the MySQL server installed on one host and you are trying to connect to the server from a different host (usually called client), you should specify the hostname or IP address of the server and the `mysql-client` package should be installed on the client. In the previous chapter, you installed both MySQL server and client packages. If you are already on the server (through SSH), you can specify `localhost`, `127.0.0.1`, or `::1`.

Second, since you are connected to the server, the next thing you need to specify is to which port you want to connect on the server. By default, MySQL runs on port `3306`. So, you should specify `3306`.

Now you know where to connect. The next obvious thing is the username and password to log in to the server. You have not created any users yet, so use the root user to connect. While installing, you would have supplied a password, use that to connect. In case you changed it, use the new password.

How to do it...

Connecting to the MySQL client can be done with any of the following commands:

```
shell> mysql -h localhost -P 3306 -u <username> -p<password>
```

```
shell> mysql --host=localhost --port=3306 --user=root --
password=<password>
```

```
shell> mysql --host localhost --port 3306 --user root --
password=<password>
```

It is highly recommended not to give the password in the command line, instead you can leave the field blank; you will be prompted for a password:

```
shell> mysql --host=localhost --port=3306 --user=root --
password
Enter Password:
```

1. The `-P` argument (in uppercase) is passed for specifying the port.
2. The `-p` argument (in lowercase) is passed for specifying the password.
3. There is no space after the `-p` argument.
4. For the password, there is no space after `=`.

By default, the host is taken as `localhost`, the port is taken as `3306`, and the user is taken as the current shell user.

1. To know the current user:

```
      shell> whoami
```

2. To disconnect, press *Ctrl* + *D* or type `exit`:

```
mysql> ^DBye
shell>
```

Or use:

```
mysql> exit;
Bye
shell>
```

3. After connecting to the `mysql` prompt, you can execute statements followed by the delimiter. The default delimiter is a semicolon (`;`):

```
mysql> SELECT 1;
+---+
| 1 |
+---+
| 1 |
+---+
1 row in set (0.00 sec)
```

2. To cancel a command, press *Ctrl* + *C* or type `\c`:

```
mysql> SELECT ^C
mysql> SELECT \c
```

Connecting to MySQL using the root user is not recommended. You can create users and restrict the user by granting appropriate privileges, which will be discussed in the Creating Users and Granting and revoking access to users sections. Till then, you can use the root user to connect to MySQL.

See also

After connecting, you might have noticed a warning:

```
Warning: Using a password on the command line interface can
be insecure.
```

To learn about the secure ways of connecting, refer to Chapter 14, *Security.*

Once you are connected to the command-line prompt, you can execute the SQL statements, which can be terminated by `;`, `\g`, or `\G`.
`;` or `\g`—output is displayed horizontally, `\G`—output is displayed vertically.

Creating databases

Well, you have installed MySQL 8.0 and connected to it. Now it is time to store some data in it, that's what the database is meant for, after all. In any **relational database management system** (**RDBMS**), data is stored in rows, which is the basic building block of the database. Rows contain columns in which we can store several set of values.

For example, if you want to store information about your customers in a database.

Here is the dataset:

```
customer id=1, first_name=Mike, last_name=Christensen
country=USA
customer id=2, first_name=Andy, last_name=Hollands,
country=Australia
customer id=3, first_name=Ravi, last_name=Vedantam,
country=India
customer id=4, first_name= Rajiv, last_name=Perera,
country=Sri Lanka
```

You should save them as rows: `(1, 'Mike', 'Christensen', 'USA')`, `(2, 'Andy', 'Hollands', 'Australia')`, `(3, 'Ravi', 'Vedantam', 'India')`, `(4, 'Rajiv', 'Perera', 'Sri Lanka')`. For this dataset, there are four rows described by three columns `(id, first_name, last_name and country)`, which are stored in a table. The number of columns that a table can hold should be defined at the time of the creation of the table, which is the major limitation of RDBMS. However, we can alter the definition of the table any time, but the full table should be rebuilt while doing so. In some cases, the table will be unavailable while doing an alter. Altering a table will be discussed in detail in Chapter 9, *Table Maintenance*.

A database is a collection of many tables, and a database server can hold many of these databases. The flow is as follows:

Database Server —> Databases —> Tables (defined by columns) —> Rows

Databases and tables are referred to as database objects. Any operation, such as creating, modifying, or deleting database objects, is called **Data Definition Language** (**DDL**).

The organization of data as a blueprint of how the database is constructed (divided into database and tables) is called a **schema**.

How to do it...

Connect to the MySQL server:

```
shell> mysql -u root -p
Enter Password:
mysql> CREATE DATABASE company;
mysql> CREATE DATABASE `my.contacts`;
```

The back tick character (`` ` ``) is used to quote identifiers, such as database and table names. You need to use it when the database name contains special characters, such as dot (`.`).

You can switch between databases:

```
mysql> USE company
mysql> USE `my.contacts`
```

Instead of switching, you can directly connect to the database you want by specifying it in the command line:

```
shell> mysql -u root -p company
```

To find which database you are connected to, use the following:

```
mysql> SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
| company    |
+------------+
1 row in set (0.00 sec)
```

To find all the databases you have access to, use:

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| company            |
| my.contacts        |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
6 rows in set (0.00 sec)
```

The database is created as a directory inside `data directory`. The default `data directory` is `/var/lib/mysql` for repository-based installation and `/usr/local/mysql/data/` for installation through binaries. To know your current `data directory`, you can execute:

```
mysql> SHOW VARIABLES LIKE 'datadir';
+---------------+------------------------+
| Variable_name | Value                  |
+---------------+------------------------+
| datadir       | /usr/local/mysql/data/ |
+---------------+------------------------+
1 row in set (0.00 sec)
```

Check the files inside `data directory` :

```
shell> sudo ls -lhtr /usr/local/mysql/data/
total 185M
-rw-r----- 1 mysql mysql   56 Jun  2 16:57 auto.cnf
-rw-r----- 1 mysql mysql  257 Jun  2 16:57
performance_sche_3.SDI
drwxr-x--- 2 mysql mysql 4.0K Jun  2 16:57
performance_schema
drwxr-x--- 2 mysql mysql 4.0K Jun  2 16:57 mysql
-rw-r----- 1 mysql mysql  242 Jun  2 16:57 sys_4.SDI
drwxr-x--- 2 mysql mysql 4.0K Jun  2 16:57 sys
-rw------- 1 mysql root  1.7K Jun  2 16:58 ca-key.pem
-rw-r--r-- 1 mysql root  1.1K Jun  2 16:58 ca.pem
-rw------- 1 mysql root  1.7K Jun  2 16:58 server-key.pem
-rw-r--r-- 1 mysql root  1.1K Jun  2 16:58 server-cert.pem
```

```
-rw------- 1 mysql root  1.7K Jun  2 16:58 client-key.pem
-rw-r--r-- 1 mysql root  1.1K Jun  2 16:58 client-cert.pem
-rw------- 1 mysql root  1.7K Jun  2 16:58 private_key.pem
-rw-r--r-- 1 mysql root   451 Jun  2 16:58 public_key.pem
-rw-r----- 1 mysql mysql 1.4K Jun  2 17:46 ib_buffer_pool
-rw-r----- 1 mysql mysql    5 Jun  2 17:46 server1.pid
-rw-r----- 1 mysql mysql  247 Jun  3 13:55 company_5.SDI
drwxr-x--- 2 mysql mysql 4.0K Jun  4 08:13 company
-rw-r----- 1 mysql mysql  12K Jun  4 18:58 server1.err
-rw-r----- 1 mysql mysql  249 Jun  5 16:17 employees_8.SDI
drwxr-x--- 2 mysql mysql 4.0K Jun  5 16:17 employees
-rw-r----- 1 mysql mysql  76M Jun  5 16:18 ibdata1
-rw-r----- 1 mysql mysql  48M Jun  5 16:18 ib_logfile1
-rw-r----- 1 mysql mysql  48M Jun  5 16:18 ib_logfile0
-rw-r----- 1 mysql mysql  12M Jun 10 10:29 ibtmp1
```

See also

You might be wondering about other files and directories, such as `information_schema` and `performance_schema`, which you have not created. `information_schema` will be discussed in the *Getting information about databases and tables* section and `performance_schema` will be discussed in Chapter 13, *Performance Tuning*, in the *Using performance_schema* section.

Creating tables

While defining columns in a table, you should mention the name of the column, datatype (integer, floating point, string, and so on), and default value (if any). MySQL supports various datatypes. Refer to the MySQL documentation for more details (https://dev.mysql.com/doc/refman/8.0/en/data-types.html). Here is an overview of all datatypes. The `JSON` datatype is a new extension, which will be discussed in Chapter 3, *Using MySQL (Advanced)*:

1. Numeric: `TINYINT`, `SMALLINT`, `MEDIUMINT`, `INT`, `BIGINT`, and `BIT`.
2. Floating numbers: `DECIMAL`, `FLOAT`, and `DOUBLE`.
3. Strings: `CHAR`, `VARCHAR`, `BINARY`, `VARBINARY`, `BLOB`, `TEXT`, `ENUM`, and `SET`.
4. Spatial datatypes are also supported. Refer to https://dev.mysql.com/doc/refman/8.0/en/spatial-extensions.html for more details.
5. The `JSON` datatype - discussed in detail in the next chapter.

You can create many tables inside a database.

How to do it...

The table contains the column definition:

```
mysql> CREATE TABLE IF NOT EXISTS `company`.`customers` (
`id` int unsigned AUTO_INCREMENT PRIMARY KEY,
`first_name` varchar(20),
`last_name` varchar(20),
`country` varchar(20)
) ENGINE=InnoDB;
```

The options are explained as follows:

- **Dot notation**: Tables can be referenced using *database name dot table name* (`database.table`). If you are connected to the database, you can simply use `customers` instead of `company.customers`.
- `IF NOT EXISTS`: If a table with the same name exists and you specify this clause, MySQL simply throws a warning that the table already exists. Otherwise, MySQL will throw an error.
- `id`: It is declared as an integer since it contains only integers. Along with that, there are two key words: `AUTO_INCREMENT` and `PRIMARY KEY`.
- `AUTO_INCREMENT`: A linearly incremental sequence is automatically generated, so you do not need to worry about assigning `id` to each row.
- `PRIMARY KEY`: Each row is identified by a `UNIQUE` column that is `NOT NULL`. Only one of these columns should be defined in a table. If a table contains an `AUTO_INCREMENT` column, it is taken as `PRIMARY KEY`.
- `first_name`, `last_name`, and `country`: They contain strings, so they are defined as `varchar`.

- **Engine**: Along with the column definition, you should mention the storage engine. Some types of storage engines include `InnoDB`, `MyISAM`, `FEDERATED`, `BLACKHOLE`, `CSV`, and `MEMORY`. Out of all the engines, `InnoDB` is the only transactional engine and it is the default engine. To learn more about transactions, refer to Chapter 5, *Transactions*.

To list all the storage engines, execute the following:

```
mysql> SHOW ENGINES\G
*************************** 1. row
***************************
      Engine: MRG_MYISAM
     Support: YES
     Comment: Collection of identical MyISAM tables
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 2. row
***************************
      Engine: FEDERATED
     Support: NO
     Comment: Federated MySQL storage engine
Transactions: NULL
          XA: NULL
  Savepoints: NULL
*************************** 3. row
***************************
      Engine: InnoDB
     Support: DEFAULT
     Comment: Supports transactions, row-level locking, and
foreign keys
Transactions: YES
          XA: YES
  Savepoints: YES
*************************** 4. row
***************************
      Engine: BLACKHOLE
     Support: YES
     Comment: /dev/null storage engine (anything you write
to it disappears)
Transactions: NO
```

```
          XA: NO
  Savepoints: NO
*************************** 5. row
***************************
      Engine: CSV
     Support: YES
     Comment: CSV storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 6. row
***************************
      Engine: MEMORY
     Support: YES
     Comment: Hash based, stored in memory, useful for
temporary tables
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 7. row
***************************
      Engine: PERFORMANCE_SCHEMA
     Support: YES
     Comment: Performance Schema
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 8. row
***************************
      Engine: ARCHIVE
     Support: YES
     Comment: Archive storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 9. row
***************************
      Engine: MyISAM
     Support: YES
     Comment: MyISAM storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
9 rows in set (0.00 sec)
```

You can create many tables in a database.

Create one more table to track the payments:

```
mysql> CREATE TABLE `company`.`payments`(
`customer_name` varchar(20) PRIMARY KEY,
`payment` float
);
```

To list all the tables, use:

```
mysql> SHOW TABLES;
+-------------------+
| Tables_in_company |
+-------------------+
| customers         |
| payments          |
+-------------------+
2 rows in set (0.00 sec)
```

To see the structure of the table, execute the following:

```
mysql> SHOW CREATE TABLE customers\G
*************************** 1. row
***************************
 Table: customers
Create Table: CREATE TABLE `customers` (
 `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
 `first_name` varchar(20) DEFAULT NULL,
 `last_name` varchar(20) DEFAULT NULL,
 `country` varchar(20) DEFAULT NULL,
 PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=utf8mb4
1 row in set (0.00 sec)
```

Or use this:

```
mysql> DESC customers;
+------------+------------------+------+-----+---------+----
------------+
| Field      | Type             | Null | Key | Default |
Extra          |
+------------+------------------+------+-----+---------+----
------------+
| id         | int(10) unsigned | NO   | PRI | NULL    |
auto_increment |
| first_name | varchar(20)      | YES  |     | NULL    |
|
| last_name  | varchar(20)      | YES  |     | NULL    |
|
| country    | varchar(20)      | YES  |     | NULL    |
|
+------------+------------------+------+-----+---------+----
------------+
4 rows in set (0.01 sec)
```

MySQL creates `.ibd` files inside the `data directory`:

```
shell> sudo ls -lhtr /usr/local/mysql/data/company
total 256K
-rw-r----- 1 mysql mysql 128K Jun 4 07:36 customers.ibd
-rw-r----- 1 mysql mysql 128K Jun 4 08:24 payments.ibd
```

Cloning table structure

You can clone the structure of one table into a new table:

```
mysql> CREATE TABLE new_customers LIKE customers;
Query OK, 0 rows affected (0.05 sec)
```

You can verify the structure of the new table:

```
mysql> SHOW CREATE TABLE new_customers\G
*************************** 1. row
***************************
       Table: new_customers
Create Table: CREATE TABLE `new_customers` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `first_name` varchar(20) DEFAULT NULL,
  `last_name` varchar(20) DEFAULT NULL,
  `country` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
1 row in set (0.00 sec)
```

See also

Refer to https://dev.mysql.com/doc/refman/8.0/en/create-table.html for many other options in `Create Table`. Partitioning a table and compressing a table will be discussed in Chapter 10, *Table Maintenance* and Chapter 11, *Managing Tablespace*, respectively.

Inserting, updating, and deleting rows

The `INSERT`, `UPDATE`, `DELETE`, and `SELECT` operations are called **Data Manipulation Language** (**DML**) statements. `INSERT`, `UPDATE`, and `DELETE` are also called write operations, or simply **write(s)**. `SELECT` is a read operation and is simply called **read(s)**.

How to do it...

Let's look at each of them in detail. I am sure you will enjoy learning this. I would suggest that you try a few things on your own as well, later. By the end of this recipe, we will also have gotten to grips with truncating tables.

Inserting

The INSERT statement is used to create new records in a table:

```
mysql> INSERT IGNORE INTO `company`.`customers`(first_name,
last_name,country)
VALUES
('Mike', 'Christensen', 'USA'),
('Andy', 'Hollands', 'Australia'),
('Ravi', 'Vedantam', 'India'),
('Rajiv', 'Perera', 'Sri Lanka');
```

Or you can explicitly mention the id column, if you want to insert the specific id:

```
mysql> INSERT IGNORE INTO `company`.`customers`(id,
first_name, last_name,country)
VALUES
(1, 'Mike', 'Christensen', 'USA'),
(2, 'Andy', 'Hollands', 'Australia'),
(3, 'Ravi', 'Vedantam', 'India'),
(4, 'Rajiv', 'Perera', 'Sri Lanka');

Query OK, 0 rows affected, 4 warnings (0.00 sec)
Records: 4 Duplicates: 4 Warnings: 4
```

IGNORE: If the row already exists and the IGNORE clause is given, the new data is ignored and the INSERT statement still succeeds in producing a warning and a number of duplicates. Otherwise, if the IGNORE clause is not given, the INSERT statement produces an error. The uniqueness of a row is identified by the primary key:

```
mysql> SHOW WARNINGS;
+---------+------+---------------------------------------+
| Level   | Code | Message                               |
+---------+------+---------------------------------------+
| Warning | 1062 | Duplicate entry '1' for key 'PRIMARY' |
| Warning | 1062 | Duplicate entry '2' for key 'PRIMARY' |
| Warning | 1062 | Duplicate entry '3' for key 'PRIMARY' |
| Warning | 1062 | Duplicate entry '4' for key 'PRIMARY' |
+---------+------+---------------------------------------+
4 rows in set (0.00 sec)
```

Updating

The UPDATE statement is used to modify the existing records in a table:

```
mysql> UPDATE customers SET first_name='Rajiv', country='UK'
WHERE id=4;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

WHERE: This is the clause used for filtering. Whatever condition(s) are issued after the WHERE clause are evaluated and the filtered rows are updated.

*The WHERE clause is **mandatory**. Failing to give it will UPDATE the whole table.*
It is recommended to do data modification in a transaction, so that you can easily rollback the changes if you find anything wrong. You can refer to Chapter 5, Transactions to learn more about transactions.

Deleting

Deleting a record can be done as follows:

```
mysql> DELETE FROM customers WHERE id=4 AND
first_name='Rajiv';
Query OK, 1 row affected (0.03 sec)
```

*The `WHERE` clause is **mandatory**. Failing to give it will `DELETE` all the rows of the table.*
It is recommended to do data modification in a transaction, so that you can easily rollback the changes if you find anything wrong.

REPLACE, INSERT, ON DUPLICATE KEY UPDATE

There are many cases where you need to handle the duplicates. The uniqueness of a row is identified by the primary key. If a row already exists, `REPLACE` simply deletes the row and inserts the new row. If a row is not there, `REPLACE` behaves as `INSERT`.

`ON DUPLICATE KEY UPDATE` is used when you want to take action if the row already exists. If you specify the `ON DUPLICATE KEY UPDATE` option and the `INSERT` statement causes a duplicate value in the `PRIMARY KEY`, MySQL performs an update to the old row based on the new values.

Suppose you want to update the previous amount whenever you get payment from the same customer and concurrently insert a new record if the customer is paying for the first time. To do this, you will define an amount column and update it whenever a new payment comes in:

```
mysql> REPLACE INTO customers VALUES
(1,'Mike','Christensen','America');
Query OK, 2 rows affected (0.03 sec)
```

You can see that two rows are affected, one duplicate row is deleted and a new row is inserted:

```
mysql> INSERT INTO payments VALUES('Mike Christensen', 200)
ON DUPLICATE KEY UPDATE payment=payment+VALUES(payment);
Query OK, 1 row affected (0.00 sec)
```

```
mysql> INSERT INTO payments VALUES('Ravi Vedantam',500) ON
DUPLICATE KEY UPDATE payment=payment+VALUES(payment);
```

```
Query OK, 1 row affected (0.01 sec)
```

When `Mike Christensen` pays $300 next time, this will update the row and add this payment to the previous payment:

```
mysql> INSERT INTO payments VALUES('Mike Christensen', 300)
ON DUPLICATE KEY UPDATE payment=payment+VALUES(payment);
Query OK, 2 rows affected (0.00 sec)
```

`VALUES` (payment): refers to the value given in the `INSERT` statement. Payment refers to the column of the table.

Truncating tables

Deleting the whole table takes lot of time, as MySQL performs operations row by row. The quickest way to delete all of the rows of a table (preserving the table structure) is to use the `TRUNCATE TABLE` statement.

Truncate is a DDL operation in MySQL, meaning once the data is truncated, it cannot be rolled back:

```
mysql> TRUNCATE TABLE customers;
Query OK, 0 rows affected (0.03 sec)
```

Loading sample data

You have created the schema (databases and tables) and some data (through `INSERT`, `UPDATE`, and `DELETE`). To explain the further chapters, more data is needed. MySQL has provided a sample `employee` database and a lot of data to play around with. In this chapter, we will discuss how to get that data and store it in our database.

How to do it...

1. Download the zipped file:

```
shell> wget
'https://codeload.github.com/datacharmer/test_db/zip/ma
ster' -O master.zip
```

2. Unzip the file:

```
shell> unzip master.zip
```

3. Load the data:

```
shell> cd test_db-master

shell> mysql -u root -p < employees.sql
mysql: [Warning] Using a password on the command line
interface can be insecure.
INFO
CREATING DATABASE STRUCTURE
INFO
storage engine: InnoDB
INFO
LOADING departments
INFO
LOADING employees
INFO
LOADING dept_emp
INFO
LOADING dept_manager
INFO
LOADING titles
INFO
LOADING salaries
data_load_time_diff
NULL
```

4. Verify the data:

```
shell> mysql -u root -p  employees -A
mysql: [Warning] Using a password on the command line
interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or
\g.
Your MySQL connection id is 35
Server version: 8.0.3-rc-log MySQL Community Server
(GPL)
Copyright (c) 2000, 2017, Oracle and/or its affiliates.
All rights reserved.
Oracle is a registered trademark of Oracle Corporation
and/or its
affiliates. Other names may be trademarks of their
respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the
current input statement.
```

```
mysql> SHOW TABLES;
+-------------------------+
| Tables_in_employees     |
+-------------------------+
| current_dept_emp        |
| departments             |
| dept_emp                |
| dept_emp_latest_date    |
| dept_manager            |
| employees               |
| salaries                |
| titles                  |
+-------------------------+
8 rows in set (0.00 sec)
```

```
mysql> DESC employees\G
*************************** 1. row
***************************
  Field: emp_no
   Type: int(11)
   Null: NO
    Key: PRI
Default: NULL
  Extra:
*************************** 2. row
***************************
```

```
  Field: birth_date
   Type: date
   Null: NO
    Key:
Default: NULL
  Extra:
*************************** 3. row
***************************
  Field: first_name
   Type: varchar(14)
   Null: NO
    Key:
Default: NULL
  Extra:
*************************** 4. row
***************************
  Field: last_name
   Type: varchar(16)
   Null: NO
    Key:
Default: NULL
  Extra:
*************************** 5. row
***************************
  Field: gender
   Type: enum('M','F')
   Null: NO
    Key:
Default: NULL
  Extra:
*************************** 6. row
***************************
  Field: hire_date
   Type: date
   Null: NO
    Key:
Default: NULL
  Extra:
6 rows in set (0.00 sec)
```

Selecting data

You have inserted and updated data in the tables. Now it is time to learn how to retrieve information from the database. In this section, we will discuss how to retrieve data from the sample `employee` database that we have created.

There are many things that you can do with `SELECT`. The most common use cases will be discussed in this section. For more details on syntax and other use cases, refer to https://dev.mysql.com/doc/refman/8.0/en/select.html.

How to do it...

Select all data from the `departments` table of the `employee` database. You can use an asterisk (`*`) to select all columns from a table. It is not recommended to use it, you should always select only the data you need:

```
mysql> SELECT * FROM departments;
+---------+--------------------+
| dept_no | dept_name          |
+---------+--------------------+
| d009    | Customer Service   |
| d005    | Development        |
| d002    | Finance            |
| d003    | Human Resources    |
| d001    | Marketing          |
| d004    | Production         |
| d006    | Quality Management |
| d008    | Research           |
| d007    | Sales              |
+---------+--------------------+
9 rows in set (0.00 sec)
```

Selecting columns

Suppose you need `emp_no` and `dept_no` from `dept_manager`:

```
mysql> SELECT emp_no, dept_no FROM dept_manager;
+--------+---------+
| emp_no | dept_no |
+--------+---------+
| 110022 | d001    |
| 110039 | d001    |
| 110085 | d002    |
| 110114 | d002    |
| 110183 | d003    |
| 110228 | d003    |
| 110303 | d004    |
| 110344 | d004    |
| 110386 | d004    |
| 110420 | d004    |
| 110511 | d005    |
| 110567 | d005    |
| 110725 | d006    |
| 110765 | d006    |
| 110800 | d006    |
| 110854 | d006    |
| 111035 | d007    |
| 111133 | d007    |
| 111400 | d008    |
| 111534 | d008    |
| 111692 | d009    |
| 111784 | d009    |
| 111877 | d009    |
| 111939 | d009    |
+--------+---------+
24 rows in set (0.00 sec)
```

Count

Find the count of employees from the `employees` table:

```
mysql> SELECT COUNT(*) FROM employees;
+----------+
| COUNT(*) |
+----------+
|   300024 |
+----------+
1 row in set (0.03 sec)
```

Filter based on condition

Find `emp_no` of employees with `first_name` as `Georgi` and `last_name` as `Facello`:

```
mysql> SELECT emp_no FROM employees WHERE
first_name='Georgi' AND last_name='Facello';
+--------+
| emp_no |
+--------+
|  10001 |
|  55649 |
+--------+
2 rows in set (0.08 sec)
```

All the filtering conditions are given through the `WHERE` clause. Except integers and floating points, everything else should be put inside quotes.

Operators

MySQL supports many operators for filtering results. Refer to https://dev.mysql.com/doc/refman/8.0/en/comparison-operators.html for a list of all the operators. We will discuss a few operators here. `LIKE` and `RLIKE` are explained in detail in the next examples:

- **Equality**: Refer to the preceding example where you have filtered using `=`.
- `IN`: Check whether a value is within a set of values.
  For example, find the count of all employees whose last name is either `Christ`, `Lamba`, or `Baba`:

```
mysql> SELECT COUNT(*) FROM employees WHERE last_name
IN ('Christ', 'Lamba', 'Baba');
+----------+
| COUNT(*) |
+----------+
|      626 |
+----------+
1 row in set (0.08 sec)
```

- `BETWEEN...AND`: Check whether a value is within a range of values.
  For example, find the number of employees who were hired in December 1986:

```
mysql> SELECT COUNT(*) FROM employees WHERE hire_date
BETWEEN '1986-12-01' AND '1986-12-31';
+----------+
| COUNT(*) |
+----------+
|     3081 |
+----------+
1 row in set (0.06 sec)
```

- `NOT`: You can simply negate the results by preceding with the `NOT` operator.
  For example, find the number of employees who were `NOT` hired in December 1986:

```
mysql> SELECT COUNT(*) FROM employees WHERE hire_date
NOT BETWEEN '1986-12-01' AND '1986-12-31';
+----------+
| COUNT(*) |
+----------+
|   296943 |
+----------+
1 row in set (0.08 sec)
```

Simple pattern matching

You can use the `LIKE` operator. Use underscore (_) for matching exactly one character. Use `%` for matching any number of characters.

- Find the count of all employees whose first name starts with `Christ`:

```
mysql> SELECT COUNT(*) FROM employees WHERE first_name
LIKE 'christ%';
+----------+
| COUNT(*) |
+----------+
|     1157 |
+----------+
1 row in set (0.06 sec)
```

- Find the count of all employees whose first name starts with `Christ` and ends with `ed`:

```
mysql> SELECT COUNT(*) FROM employees WHERE first_name
LIKE 'christ%ed';
+----------+
| COUNT(*) |
+----------+
|      228 |
+----------+
1 row in set (0.06 sec)
```

- Find the count of all employees whose first name contains `sri`:

```
mysql> SELECT COUNT(*) FROM employees WHERE first_name
LIKE '%sri%';
+----------+
| COUNT(*) |
+----------+
|      253 |
+----------+
1 row in set (0.08 sec)
```

- Find the count of all employees whose first name ends with `er`:

```
mysql> SELECT COUNT(*) FROM employees WHERE first_name
LIKE '%er';
+----------+
| COUNT(*) |
+----------+
|     5388 |
+----------+
1 row in set (0.08 sec)
```

- Find the count of all employees whose first name starts with any two characters followed by `ka` and then followed by any number of characters:

```
mysql> SELECT COUNT(*) FROM employees WHERE first_name
LIKE '__ka%';
+----------+
| COUNT(*) |
+----------+
|     1918 |
+----------+
1 row in set (0.06 sec)
```

Regular expressions

You can use regular expressions in the `WHERE` clause by using the `RLIKE` or `REGEXP` operators. There are many ways to use `REGEXP`, refer to https://dev.mysql.com/doc/refman/8.0/en/regexp.html for more examples:

Expression	Description
`*`	Zero or more repetitions
`+`	One or more repetitions
`?`	Optional character
`.`	Any character
`\.`	Period
`^`	Starts with
`$`	Ends with
`[abc]`	Only *a*, *b*, or *c*
`[^abc]`	Neither *a*, *b*, nor *c*

`[a-z]`	Characters a to z
`[0-9]`	Numbers 0 to 9
`^...$`	Starts and ends
`\d`	Any digit
`\D`	Any non-digit character
`\s`	Any whitespace
`\S`	Any non-whitespace character
`\w`	Any alphanumeric character
`\W`	Any non-alphanumeric character
`{m}`	*m* repetitions
`{m,n}`	*m* to *n* repetitions

- Find the count of all employees whose first name starts with `christ`:

```
mysql> SELECT COUNT(*) FROM employees WHERE first_name
RLIKE '^christ';
+----------+
| COUNT(*) |
+----------+
|     1157 |
+----------+
1 row in set (0.18 sec)
```

- Find the count of all employees whose last name ends with `ba`:

```
mysql> SELECT COUNT(*) FROM employees WHERE last_name
REGEXP 'ba$';
+----------+
| COUNT(*) |
+----------+
|     1008 |
+----------+
1 row in set (0.15 sec)
```

- Find the count of all employees whose last name does not contain vowels (a, e, i, o, or u):

```
mysql> SELECT COUNT(*) FROM employees WHERE last_name
NOT REGEXP '[aeiou]';
+----------+
| COUNT(*) |
+----------+
|      148 |
+----------+
1 row in set (0.11 sec)
```

Limiting results

Select the names of any 10 employees whose `hire_date` is before 1986. You can get this by using the `LIMIT` clause at the end of the statement:

```
mysql> SELECT first_name, last_name FROM employees WHERE
hire_date < '1986-01-01' LIMIT 10;
+------------+------------+
| first_name | last_name  |
+------------+------------+
| Bezalel    | Simmel     |
| Sumant     | Peac       |
| Eberhardt  | Terkki     |
| Otmar      | Herbst     |
| Florian    | Syrotiuk   |
| Tse        | Herber     |
| Udi        | Jansch     |
| Reuven     | Garigliano |
| Erez       | Ritzmann   |
| Premal     | Baek       |
+------------+------------+
10 rows in set (0.00 sec)
```

Using the table alias

By default, whatever column you have given in the `SELECT` clause will appear in the results. In the previous examples, you have found out the count, but it is displayed as `COUNT(*)`. You can change it by using the `AS` alias:

```
mysql> SELECT COUNT(*) AS count FROM employees WHERE
hire_date BETWEEN '1986-12-01' AND '1986-12-31';
+-------+
| count |
+-------+
|  3081 |
+-------+
1 row in set (0.06 sec)
```

Sorting results

You can order the result based on the column or aliased column. You can be specify `DESC` for descending order or `ASC` for ascending. By default, ordering will be ascending. You can combine the `LIMIT` clause with `ORDER BY` to limit the results.

How to do it...

Find the employee IDs of the first five top-paid employees.

```
mysql> SELECT emp_no,salary FROM salaries ORDER BY salary
DESC LIMIT 5;
+--------+--------+
| emp_no | salary |
+--------+--------+
|  43624 | 158220 |
|  43624 | 157821 |
| 254466 | 156286 |
|  47978 | 155709 |
| 253939 | 155513 |
+--------+--------+
5 rows in set (0.74 sec)
```

Instead of specifying the column name, you can also mention the position of the column in the `SELECT` statement. For example, you are selecting the salary at the second position in the `SELECT` statement. So, you can specify `ORDER BY 2`:

```
mysql> SELECT emp_no,salary FROM salaries ORDER BY 2 DESC
LIMIT 5;
+--------+--------+
| emp_no | salary |
+--------+--------+
|  43624 | 158220 |
|  43624 | 157821 |
| 254466 | 156286 |
|  47978 | 155709 |
| 253939 | 155513 |
+--------+--------+
5 rows in set (0.78 sec)
```

Grouping results (aggregate functions)

You can group the results using the `GROUP BY` clause on a column and then use `AGGREGATE` functions, such as `COUNT`, `MAX`, `MIN`, and `AVERAGE`. You can also use the function on a column in a group by clause. See the `SUM` example where you will use the `YEAR()` function.

How to do it...

Each of the previously-mentioned aggregate functions will be introduced to you here in detail.

COUNT

1. Find the count of male and female employees:

```
mysql> SELECT gender, COUNT(*) AS count FROM employees
GROUP BY gender;
+--------+--------+
| gender | count  |
+--------+--------+
| M      | 179973 |
| F      | 120051 |
+--------+--------+
2 rows in set (0.14 sec)
```

2. You want to find the 10 most common first names of the employees. You can use `GROUP BY first_name` to group all the first names, then `COUNT(first_name)` to find the count inside the group, and finally the `ORDER BY` count to sort the results. `LIMIT` these results to the top 10:

```
mysql> SELECT first_name, COUNT(first_name) AS count
FROM employees GROUP BY first_name ORDER BY count DESC
LIMIT 10;
+-------------+-------+
| first_name  | count |
+-------------+-------+
| Shahab      |   295 |
| Tetsushi    |   291 |
| Elgin       |   279 |
| Anyuan      |   278 |
| Huican      |   276 |
| Make        |   275 |
| Panayotis   |   272 |
| Sreekrishna |   272 |
| Hatem       |   271 |
| Giri        |   270 |
+-------------+-------+
10 rows in set (0.21 sec)
```

SUM

Find the sum of the salaries given to employees in each year and sort the results by salary. The `YEAR()` function returns the `YEAR` of the given date:

```
mysql> SELECT '2017-06-12', YEAR('2017-06-12');
+------------+--------------------+
| 2017-06-12 | YEAR('2017-06-12') |
+------------+--------------------+
| 2017-06-12 |               2017 |
+------------+--------------------+
1 row in set (0.00 sec)
```

```
mysql>  SELECT YEAR(from_date), SUM(salary) AS sum FROM
salaries GROUP BY YEAR(from_date) ORDER BY sum DESC;
+-----------------+-------------+
| YEAR(from_date) | sum         |
+-----------------+-------------+
|            2000 | 17535667603 |
|            2001 | 17507737308 |
|            1999 | 17360258862 |
|            1998 | 16220495471 |
|            1997 | 15056011781 |
|            1996 | 13888587737 |
|            1995 | 12638817464 |
|            1994 | 11429450113 |
|            2002 | 10243347616 |
|            1993 | 10215059054 |
|            1992 |  9027872610 |
|            1991 |  7798804412 |
|            1990 |  6626146391 |
|            1989 |  5454260439 |
|            1988 |  4295598688 |
|            1987 |  3156881054 |
|            1986 |  2052895941 |
|            1985 |   972864875 |
+-----------------+-------------+
18 rows in set (1.47 sec)
```

AVERAGE

Find the 10 employees with the highest average salaries:

```
mysql>  SELECT emp_no, AVG(salary) AS avg FROM salaries
GROUP BY emp_no ORDER BY avg DESC LIMIT 10;
+--------+-------------+
| emp_no | avg         |
+--------+-------------+
| 109334 | 141835.3333 |
| 205000 | 141064.6364 |
|  43624 | 138492.9444 |
| 493158 | 138312.8750 |
|  37558 | 138215.8571 |
| 276633 | 136711.7333 |
| 238117 | 136026.2000 |
|  46439 | 135747.7333 |
| 254466 | 135541.0625 |
| 253939 | 135042.2500 |
+--------+-------------+
10 rows in set (0.91 sec
```

DISTINCT

You can use the `DISTINCT` clause to filter the distinct entries in a table:

```
mysql> SELECT DISTINCT title FROM titles;
+--------------------+
| title              |
+--------------------+
| Senior Engineer    |
| Staff              |
| Engineer           |
| Senior Staff       |
| Assistant Engineer |
| Technique Leader   |
| Manager            |
+--------------------+
7 rows in set (0.30 sec)
```

Filtering using HAVING

You can filter results of the `GROUP BY` clause by adding the `HAVING` clause.

For example, find the employees with an average salary of more than 140,000:

```
mysql>  SELECT emp_no, AVG(salary) AS avg FROM salaries
GROUP BY emp_no HAVING avg > 140000 ORDER BY avg DESC;
+--------+-------------+
| emp_no | avg         |
+--------+-------------+
| 109334 | 141835.3333 |
| 205000 | 141064.6364 |
+--------+-------------+
2 rows in set (0.80 sec)
```

See also

There are many other aggregate functions, refer to https://dev.mysql.com/doc/refman/8.0/en/group-by-functions.html for more information.

Creating users

So far, you have used only the root user to connect to MySQL and execute statements. The root user should never be used while accessing MySQL, except for administrative tasks from `localhost`. You should create users, restrict the access, restrict the resource usage, and so on. For creating new users, you should have the `CREATE USER` privilege that will be discussed in the next section. During the initial set up, you can use the root user to create other users.

How to do it...

Connect to mysql using the root user and execute `CREATE USER` command to create new users.

```
mysql> CREATE USER IF NOT EXISTS
'company_read_only'@'localhost'
IDENTIFIED WITH mysql_native_password
BY 'company_pass'
WITH MAX_QUERIES_PER_HOUR 500
MAX_UPDATES_PER_HOUR 100;
```

You might get the following error if the password is not strong.

```
ERROR 1819 (HY000): Your password does not satisfy the
current policy requirements
```

The preceding statement will create users with:

- `* Username`: `company_read_only`.
- `* access only from`: `localhost`.
- You can restrict the access to the IP range. For example: `10.148.%.%`. By giving `%`, the user can access from any host.
- `* password`: `company_pass`.
- `* using mysql_native_password` (default) authentication.
- You can also specify any pluggable authentication, such as `sha256_password`, `LDAP`, or Kerberos.
- The `* maximum number of queries` the user can execute in an hour is 500.
- The `* maximum number of updates` the user can execute in an hour is 100.

When a client connects to the MySQL server, it undergoes two stages:

1. Access control—connection verification
2. Access control—request verification

During the connection verification, the server identifies the connection by the username and the hostname from which it is connected. The server invokes the authentication plugin for the user and verifies the password. It also checks whether the user is locked or not.

During the request verification stage, the server checks whether the user has sufficient privileges for each operation.

In the preceding statement, you have to give the password in clear text, which can be recorded in the command history file, `$HOME/.mysql_history`. To avoid that, you can compute the hash on your local server and directly specify the hashed string. The syntax for it is the same, except `mysql_native_password BY 'company_pass'` changes to `mysql_native_password AS 'hashed_string'`:

```
mysql> SELECT PASSWORD('company_pass');
+-------------------------------------------+
|PASSWORD('company_pass')                   |
+-------------------------------------------+
| *EBD9E3BFD1489CA1EB0D2B4F29F6665F321E8C18 |
+-------------------------------------------+
1 row in set, 1 warning (0.00 sec)
```

```
mysql> CREATE USER IF NOT EXISTS
'company_read_only'@'localhost'
IDENTIFIED WITH mysql_native_password
```

```
AS '*EBD9E3BFD1489CA1EB0D2B4F29F6665F321E8C18'
WITH MAX_QUERIES_PER_HOUR 500
```

```
MAX_UPDATES_PER_HOUR 100;
```

You can directly create users by granting privileges. Refer to the next section on how to grant privileges. However, MySQL will deprecate this feature in the next release.

See also

Refer to https://dev.mysql.com/doc/refman/8.0/en/create-user.html for more options on creating users. More secure options, such as SSL, using other authentication methods will be discussed in Chapter 14, *Security*.

Granting and revoking access to users

You can restrict the user to access specific databases or tables and also only specific operations, such as `SELECT`, `INSERT`, and `UPDATE`. For granting privileges to other users, you should have the `GRANT` privilege.

How to do it...

During the initial setup, you can use the root user to grant privileges. You can also create an administrative account to manage the users.

Granting privileges

- Grant the `READ ONLY(SELECT)` privileges to the `company_read_only` user:

```
mysql> GRANT SELECT ON company.* TO
'company_read_only'@'localhost';
Query OK, 0 rows affected (0.06 sec)
```

  The asterisk (`*`) represents all tables inside the database.

- Grant the `INSERT` privilege to the new `company_insert_only` user:

```
mysql> GRANT INSERT ON company.* TO
'company_insert_only'@'localhost' IDENTIFIED BY 'xxxx';
Query OK, 0 rows affected, 1 warning (0.05 sec)
```

```
mysql> SHOW WARNINGS\G
*************************** 1. row
***************************
  Level: Warning
   Code: 1287
Message: Using GRANT for creating new user is
deprecated and will be removed in future release.
Create new user with CREATE USER statement.
1 row in set (0.00 sec)
```

- Grant the `WRITE` privileges to the new `company_write` user:

```
mysql> GRANT INSERT, DELETE, UPDATE ON company.* TO
'company_write'@'%' IDENTIFIED WITH
mysql_native_password AS
'*EBD9E3BFD1489CA1EB0D2B4F29F6665F321E8C18';
Query OK, 0 rows affected, 1 warning (0.04 sec)
```

- Restrict to a specific table. Restrict the `employees_read_only` user to `SELECT` only from the `employees` table:

```
mysql> GRANT SELECT ON employees.employees TO
'employees_read_only'@'%' IDENTIFIED WITH
mysql_native_password AS
'*EBD9E3BFD1489CA1EB0D2B4F29F6665F321E8C18';
Query OK, 0 rows affected, 1 warning (0.03 sec)
```

- You can further restrict to specific columns. Restrict the `employees_ro` user to the `first_name` and `last_name` columns of the `employees` table:

```
mysql> GRANT SELECT(first_name,last_name)  ON
employees.employees TO 'employees_ro'@'%' IDENTIFIED
WITH mysql_native_password AS
'*EBD9E3BFD1489CA1EB0D2B4F29F6665F321E8C18';
Query OK, 0 rows affected, 1 warning (0.06 sec)
```

- Extending grants. You can extend the grants by executing the new grant. Extend the privilege to the `employees_col_ro` user to access the salary of the `salaries` table:

```
mysql> GRANT SELECT(salary) ON employees.salaries TO
'employees_ro'@'%';
Query OK, 0 rows affected (0.00 sec)
```

- Creating the `SUPER` user. You need an administrative account to manage the server. `ALL` signifies all privileges expect the `GRANT` privilege:

```
mysql> CREATE USER 'dbadmin'@'%' IDENTIFIED WITH
mysql_native_password BY 'DB@dm1n';
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> GRANT ALL ON *.* TO 'dbadmin'@'%';
Query OK, 0 rows affected (0.01 sec)
```

- Granting the `GRANT` privilege. The user should have the `GRANT OPTION` privilege to grant privileges to other users. You can extend the `GRANT` privilege to the `dbadmin` super user:

```
mysql> GRANT GRANT OPTION ON *.* TO 'dbadmin'@'%';
Query OK, 0 rows affected (0.03 sec)
```

Refer to https://dev.mysql.com/doc/refman/8.0/en/grant.html for more privilege types.

Checking grants

You can check all the user's grants. Check grants for the `employee_col_ro` user:

```
mysql> SHOW GRANTS FOR 'employees_ro'@'%'\G
*************************** 1. row
***************************
Grants for employees_ro@%: GRANT USAGE ON *.* TO
`employees_ro`@`%`
*************************** 2. row
***************************
Grants for employees_ro@%: GRANT SELECT (`first_name`,
`last_name`) ON `employees`.`employees` TO
`employees_ro`@`%`
*************************** 3. row
***************************
Grants for employees_ro@%: GRANT SELECT (`salary`) ON
`employees`.`salaries` TO `employees_ro`@`%`
```

Check grants for the `dbadmin` user. You can see all the grants that are available to the `dbadmin` user:

```
mysql> SHOW GRANTS FOR 'dbadmin'@'%'\G
*************************** 1. row
***************************
Grants for dbadmin@%: GRANT SELECT, INSERT, UPDATE, DELETE,
CREATE, DROP, RELOAD, SHUTDOWN, PROCESS, FILE, REFERENCES,
INDEX, ALTER, SHOW DATABASES, SUPER, CREATE TEMPORARY
TABLES, LOCK TABLES, EXECUTE, REPLICATION SLAVE, REPLICATION
CLIENT, CREATE VIEW, SHOW VIEW, CREATE ROUTINE, ALTER
ROUTINE, CREATE USER, EVENT, TRIGGER, CREATE TABLESPACE,
CREATE ROLE, DROP ROLE ON *.* TO `dbadmin`@`%` WITH GRANT
OPTION
*************************** 2. row
***************************
Grants for dbadmin@%: GRANT
BINLOG_ADMIN,CONNECTION_ADMIN,ENCRYPTION_KEY_ADMIN,GROUP_REP
LICATION_ADMIN,REPLICATION_SLAVE_ADMIN,ROLE_ADMIN,SET_USER_I
```

```
D,SYSTEM_VARIABLES_ADMIN ON *.* TO `dbadmin`@`%`
2 rows in set (0.00 sec)
```

Revoking grants

Revoking grants has the same syntax as creating grants. You grant a privilege `TO` the user and revoke a privilege `FROM` the user.

- Revoke the `DELETE` access from the `'company_write'@'%'` user:

```
mysql> REVOKE DELETE ON company.* FROM
'company_write'@'%';
Query OK, 0 rows affected (0.04 sec)
```

- Revoke the access to the salary column from the `employee_ro` user:

```
mysql> REVOKE SELECT(salary) ON employees.salaries FROM
'employees_ro'@'%';
Query OK, 0 rows affected (0.03 sec)
```

Modifying the mysql.user table

All the user information, along with privileges, is stored in the `mysql.user` table. If you have the privilege to access the `mysql.user` table, you can directly modify the `mysql.user` table to create users and grant privileges.

If you modify the grant tables indirectly, using account-management statements such as `GRANT`, `REVOKE`, `SET PASSWORD`, or `RENAME USER`, the server notices these changes and loads the grant tables into memory again immediately.

If you modify the grant tables directly, using statements such as `INSERT`, `UPDATE`, or `DELETE`, your changes have no effect on privilege checking until you either restart the server or tell it to reload the tables. If you change the grant tables directly but forget to reload them, your changes have no effect until you restart the server.

The reloading of the `GRANT` tables can be done by issuing a `FLUSH PRIVILEGES` statement.

Query the `mysql.user` table to find out all the entries for the `dbadmin` user:

```
mysql> SELECT * FROM mysql.user WHERE user='dbadmin'\G
*************************** 1. row
***************************
                  Host: %
                  User: dbadmin
           Select_priv: Y
           Insert_priv: Y
           Update_priv: Y
           Delete_priv: Y
           Create_priv: Y
```

```
             Drop_priv: Y
           Reload_priv: Y
         Shutdown_priv: Y
          Process_priv: Y
             File_priv: Y
            Grant_priv: Y
       References_priv: Y
            Index_priv: Y
            Alter_priv: Y
          Show_db_priv: Y
            Super_priv: Y
 Create_tmp_table_priv: Y
      Lock_tables_priv: Y
          Execute_priv: Y
       Repl_slave_priv: Y
      Repl_client_priv: Y
      Create_view_priv: Y
        Show_view_priv: Y
   Create_routine_priv: Y
    Alter_routine_priv: Y
      Create_user_priv: Y
            Event_priv: Y
          Trigger_priv: Y
Create_tablespace_priv: Y
              ssl_type:
            ssl_cipher:
           x509_issuer:
          x509_subject:
         max_questions: 0
           max_updates: 0
       max_connections: 0
  max_user_connections: 0
                plugin: mysql_native_password
 authentication_string:
*AB7018ADD9CB4EDBEB680BB3F820479E4CE815D2
      password_expired: N
 password_last_changed: 2017-06-10 16:24:03
     password_lifetime: NULL
        account_locked: N
      Create_role_priv: Y
        Drop_role_priv: Y
1 row in set (0.00 sec)
```

You can see that the `dbadmin` user can access the database from any host (%). You can restrict them to `localhost` just by updating the `mysql.user` table and reloading the grant tables:

```
mysql> UPDATE mysql.user SET host='localhost' WHERE
user='dbadmin';
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

```
mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)
```

Setting password expiry for users

You can expire the passwords of users for a specific interval; after this, they need to change their password.

When an application developer asks for database access, you can create the account with a default password and then set it to expire. You can share the password with the developers, then they have to change the password to continue using MySQL.

All the accounts are created with a password expiry equal to the `default_password_lifetime` variable, which is disabled by default:

- Create a user with an expired password. When the developer logs in for the first time and tries to execute any statement, `ERROR 1820 (HY000):` is thrown. The password must be reset using the `ALTER USER` statement before executing this statement:

```
mysql> CREATE USER 'developer'@'%' IDENTIFIED WITH
mysql_native_password AS
'*EBD9E3BFD1489CA1EB0D2B4F29F6665F321E8C18' PASSWORD
EXPIRE;
Query OK, 0 rows affected (0.04 sec
```

```
shell> mysql -u developer -pcompany_pass
mysql: [Warning] Using a password on the command line
interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or
\g.
Your MySQL connection id is 31
Server version: 8.0.3-rc-log

Copyright (c) 2000, 2017, Oracle and/or its affiliates.
All rights reserved.
```

```
Oracle is a registered trademark of Oracle Corporation
and/or its
affiliates. Other names may be trademarks of their
respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the
current input statement.

mysql> SHOW DATABASES;
ERROR 1820 (HY000): You must reset your password using
ALTER USER statement before executing this statement.
```

The developer has to change their password using the following command:

```
mysql> ALTER USER 'developer'@'%' IDENTIFIED WITH
mysql_native_password BY 'new_company_pass';
Query OK, 0 rows affected (0.03 sec)
```

- Manually expire the existing user:

```
mysql> ALTER USER 'developer'@'%' PASSWORD EXPIRE;
Query OK, 0 rows affected (0.06 sec)
```

- Require the password to be changed every 180 days:

```
mysql> ALTER USER 'developer'@'%' PASSWORD EXPIRE
INTERVAL 90 DAY;
Query OK, 0 rows affected (0.04 sec)
```

Locking users

If you find any issues with the account, you can lock it. MySQL supports locking while using `CREATE USER` or `ALTER USER`.

Lock the account by adding the `ACCOUNT LOCK` clause to the `ALTER USER` statement:

```
mysql> ALTER USER 'developer'@'%' ACCOUNT LOCK;
Query OK, 0 rows affected (0.05 sec)
```

The developer will get an error saying that the account is locked:

```
shell> mysql -u developer -pnew_company_pass
mysql: [Warning] Using a password on the command line
interface can be insecure.
ERROR 3118 (HY000): Access denied for user
'developer'@'localhost'. Account is locked.
```

You can unlock the account after confirming:

```
mysql> ALTER USER 'developer'@'%' ACCOUNT UNLOCK;
Query OK, 0 rows affected (0.00 sec)
```

Creating roles for users

A MySQL role is a named collection of privileges. Like user accounts, roles can have privileges granted to and revoked from them. A user account can be granted roles, which grants to the account the role privileges. Earlier, you created separate users for reads, writes, and administration. For write privilege, you have granted `INSERT`, `DELETE`, and `UPDATE` to the user. Instead, you can grant those privileges to a role and then assign the user to that role. By this way, you can avoid granting privileges individually to possibly many user accounts.

- Creating roles:

```
mysql> CREATE ROLE 'app_read_only', 'app_writes',
'app_developer';
Query OK, 0 rows affected (0.01 sec)
```

- Assigning privileges to the roles using the `GRANT` statement:

```
mysql> GRANT SELECT ON employees.* TO 'app_read_only';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> GRANT INSERT, UPDATE, DELETE ON employees.* TO
'app_writes';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> GRANT ALL ON employees.* TO 'app_developer';
Query OK, 0 rows affected (0.04 sec)
```

- Creating users. If you do not specify any host, `%` will be taken:

```
mysql> CREATE user emp_read_only IDENTIFIED BY
'emp_pass';
Query OK, 0 rows affected (0.06 sec)
```

```
mysql> CREATE user emp_writes IDENTIFIED BY 'emp_pass';
Query OK, 0 rows affected (0.04 sec)
```

```
mysql> CREATE user emp_developer IDENTIFIED BY
'emp_pass';
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> CREATE user emp_read_write IDENTIFIED BY
'emp_pass';
Query OK, 0 rows affected (0.00 sec)
```

- Assigning the roles to the users using the `GRANT` statement. You can assign multiple roles to a user.
  For example, you can assign both read and write access to the `emp_read_write` user:

```
mysql> GRANT 'app_read_only' TO 'emp_read_only'@'%';
Query OK, 0 rows affected (0.04 sec)
```

```
mysql> GRANT 'app_writes' TO 'emp_writes'@'%';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> GRANT 'app_developer' TO 'emp_developer'@'%';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> GRANT 'app_read_only', 'app_writes' TO
'emp_read_write'@'%';
Query OK, 0 rows affected (0.05 sec)
```

As a security measure, avoid using `%` and restrict the access to the IP where the application is deployed.

Selecting data into a file and table

You can save the output into a file using the `SELECT INTO OUTFILE` statement.

You can specify the column and line delimiters, and later you can import the data into other data platforms.

How to do it...

You can save the output destination as a file or a table.

Saving as a file

- To save the output into a file, you need the `FILE` privilege. `FILE` is a global privilege, which means you cannot restrict it for a particular database. However, you can restrict what the user selects:

```
mysql> GRANT SELECT ON employees.* TO
'user_ro_file'@'%' IDENTIFIED  WITH
mysql_native_password AS
'*EBD9E3BFD1489CA1EB0D2B4F29F6665F321E8C18';
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> GRANT FILE ON *.* TO 'user_ro_file'@'%'
IDENTIFIED  WITH mysql_native_password AS
'*EBD9E3BFD1489CA1EB0D2B4F29F6665F321E8C18';
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

- On Ubuntu, by default, MySQL will not allow you to write to file. You should set `secure_file_priv` in the config file and restart MySQL. You will learn more on config in Chapter 4, *Configuring MySQL*. On CentOS, Red Hat, `secure_file_priv` is set to `/var/lib/mysql-files`, which means all the files will be saved in that directory.
- For now, enable like this. Open the config file and add `secure_file_priv = /var/lib/mysql`:

```
shell> sudo vi /etc/mysql/mysql.conf.d/mysqld.cnf
```

- Restart the MySQL server:

```
shell> sudo systemctl restart mysql
```

The following statement will save the output into a CSV format:

```
mysql> SELECT first_name, last_name INTO OUTFILE
'result.csv'
       FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
       LINES TERMINATED BY '\n'
       FROM employees WHERE hire_date<'1986-01-01' LIMIT 10;
Query OK, 10 rows affected (0.00 sec)
```

You can check the output of the file, which will be created in the path specified by `{secure_file_priv}/{database_name}`, it is `/var/lib/mysql/employees/` in this case. The statement will fail if the file already exists, so you need to give a unique name every time you execute or move the file to a different location:

```
shell> sudo cat /var/lib/mysql/employees/result.csv
"Bezalel","Simmel"
"Sumant","Peac"
"Eberhardt","Terkki"
"Otmar","Herbst"
"Florian","Syrotiuk"
"Tse","Herber"
"Udi","Jansch"
"Reuven","Garigliano"
"Erez","Ritzmann"
"Premal","Baek"
```

Saving as a table

You can save the results of a `SELECT` statement into a table. Even if the table does not exist, you can use `CREATE` and `SELECT` to create the table and load the data. If the table already exists, you can use `INSERT` and `SELECT` to load the data.

You can save the titles into a new `titles_only` table:

```
mysql> CREATE TABLE titles_only AS SELECT DISTINCT title
FROM titles;
Query OK, 7 rows affected (0.50 sec)
Records: 7  Duplicates: 0  Warnings: 0
```

If the table already exists, you can use the `INSERT INTO SELECT` statement:

```
mysql> INSERT INTO titles_only SELECT DISTINCT title FROM
titles;
Query OK, 7 rows affected (0.46 sec)
Records: 7  Duplicates: 0  Warnings: 0
```

To avoid duplicates, you can use `INSERT IGNORE`. However, in this case, there is no `PRIMARY KEY` on the `titles_only` table. So the `IGNORE` clause does not make any difference.

Loading data into a table

The way you can dump a table data into a file, you can do vice-versa, that is, load the data from the file into a table. This is widely used for loading bulk data and is a super fast way to load data into tables. You can specify the column delimiters to load the data into respective columns. You should have the `FILE` privilege and the `INSERT` privilege on the table.

How to do it...

Earlier, you have saved `first_name` and `last_name` to a file. You can use the same file to load the data into another table. Before loading, you should create the table. If the table already exists, you can directly load. The columns of the table should match the fields of the file.

Create a table to hold the data:

```
mysql> CREATE TABLE employee_names (
       `first_name` varchar(14) NOT NULL,
       `last_name` varchar(16) NOT NULL
       ) ENGINE=InnoDB;
Query OK, 0 rows affected (0.07 sec)
```

Make sure that the file is present:

```
shell> sudo ls -lhtr /var/lib/mysql/employees/result.csv
-rw-rw-rw- 1 mysql mysql 180 Jun 10 14:53
/var/lib/mysql/employees/result.csv
```

Load the data using the `LOAD DATA INFILE` statement:

```
mysql> LOAD DATA INFILE 'result.csv' INTO TABLE
employee_names
       FIELDS TERMINATED BY ','
       OPTIONALLY ENCLOSED BY '"'
       LINES TERMINATED BY '\n';
Query OK, 10 rows affected (0.01 sec)
Records: 10  Deleted: 0  Skipped: 0  Warnings: 0
```

The file can be given as a full path name to specify its exact location. If given as a relative path name, the name is interpreted relative to the directory in which the client program was started.

- If the file contains any headers you want to ignore, specify `IGNORE n LINES`:

```
mysql> LOAD DATA INFILE 'result.csv' INTO TABLE
employee_names
       FIELDS TERMINATED BY ','
       OPTIONALLY ENCLOSED BY '"'
       LINES TERMINATED BY '\n'
       IGNORE 1 LINES;
```

- You can specify `REPLACE` or `IGNORE` to deal with duplicates:

```
mysql> LOAD DATA INFILE 'result.csv' REPLACE INTO TABLE
employee_names FIELDS TERMINATED BY ','OPTIONALLY
ENCLOSED BY '"' LINES TERMINATED BY '\n';
Query OK, 10 rows affected (0.01 sec)
Records: 10  Deleted: 0  Skipped: 0  Warnings: 0

mysql> LOAD DATA INFILE 'result.csv' IGNORE INTO TABLE
employee_names FIELDS TERMINATED BY ','OPTIONALLY
ENCLOSED BY '"' LINES TERMINATED BY '\n';
Query OK, 10 rows affected (0.06 sec)
Records: 10  Deleted: 0  Skipped: 0  Warnings: 0
```

- MySQL assumes that the file you want to load is available on the server. If you are connected to the server from a remote client machine, you can specify `LOCAL` to load the file located on the client. The local file will be copied from the client to the server. The file is saved in the standard temporary location of the server. In Linux machines, it is `/tmp`:

```
mysql> LOAD DATA LOCAL INFILE 'result.csv' IGNORE INTO
TABLE employee_names FIELDS TERMINATED BY ','OPTIONALLY
ENCLOSED BY '"' LINES TERMINATED BY '\n';
```

Joining tables

So far you have looked at inserting and retrieving data from a single table. In this section, we will discuss how to combine two or more tables to retrieve the results.

A perfect example is that you want to find the employee name and department number of a employee with `emp_no: 110022`:

- The department number and name are stored in the `departments` table
- The employee number and other details, such as `first_name` and `last_name`, are stored in the `employees` table
- The mapping of employee and department is stored in the `dept_manager` table

If you do not want to use `JOIN`, you can do this:

1. Find the employee name with `emp_no` as `110022` from the `employee` table:

```
mysql> SELECT emp.emp_no, emp.first_name, emp.last_name
FROM employees AS emp
WHERE  emp.emp_no=110022;
+--------+------------+------------+
| emp_no | first_name | last_name  |
+--------+------------+------------+
| 110022 | Margareta  | Markovitch |
+--------+------------+------------+
1 row in set (0.00 sec)
```

2. Find the department number from the `departments` table:

```
mysql> SELECT dept_no FROM dept_manager AS dept_mgr
WHERE dept_mgr.emp_no=110022;
+---------+
| dept_no |
+---------+
| d001    |
+---------+
1 row in set (0.00 sec)
```

3. Find the department name from the `departments` table:

```
mysql> SELECT dept_name FROM departments dept WHERE
dept.dept_no='d001';
+-----------+
| dept_name |
+-----------+
| Marketing |
+-----------+
1 row in set (0.00 sec)
```

How to do it...

To avoid look ups on three different tables using three statements, you can use `JOIN` to club them. The important thing to note here is to join two tables, you should have one, or more, common column to join. You can join employees and `dept_manager` based on `emp_no`, they both have the `emp_no` column. Though the names don't need to match, you should figure out the column on which you can join. Similarly, `dept_mgr` and `departments` have `dept_no` as a common column.

Like a column alias, you can give table an alias and refer columns of that table using an alias. For example, you can give employees an alias using `FROM employees AS emp` and refer columns of the `employees` table using dot notation, such as `emp.emp_no`:

```
mysql> SELECT
    emp.emp_no,
    emp.first_name,
    emp.last_name,
    dept.dept_name
FROM
    employees AS emp
JOIN dept_manager AS dept_mgr
    ON emp.emp_no=dept_mgr.emp_no AND emp.emp_no=110022
JOIN departments AS dept
    ON dept_mgr.dept_no=dept.dept_no;
+--------+------------+------------+-----------+
| emp_no | first_name | last_name  | dept_name |
+--------+------------+------------+-----------+
| 110022 | Margareta  | Markovitch | Marketing |
+--------+------------+------------+-----------+
1 row in set (0.00 sec)
```

Let's look at another example—you want to find out the average salary for each department. For this you can use the `AVG` function and group by `dept_no`. To find out the department name, you can join the results with the `departments` table on `dept_no`:

```
mysql> SELECT
    dept_name,
    AVG(salary) AS avg_salary
FROM
    salaries
JOIN dept_emp
    ON salaries.emp_no=dept_emp.emp_no
JOIN departments
    ON dept_emp.dept_no=departments.dept_no
GROUP BY
    dept_emp.dept_no
ORDER BY
    avg_salary
DESC;
+--------------------+------------+
| dept_name          | avg_salary |
+--------------------+------------+
| Sales              | 80667.6058 |
| Marketing          | 71913.2000 |
| Finance            | 70489.3649 |
| Research           | 59665.1817 |
| Production         | 59605.4825 |
| Development        | 59478.9012 |
| Customer Service   | 58770.3665 |
| Quality Management | 57251.2719 |
| Human Resources    | 55574.8794 |
+--------------------+------------+
9 rows in set (8.29 sec)
```

Identifying Duplicates using SELF JOIN

You want to find the duplicate rows in a table for specific columns. For example, you want to find out which employees have the same `first_name`, same `last_name`, same `gender`, and same `hire_date`. In that case, you can join the `employees` table with itself while specifying the columns where you want to find duplicates in the `JOIN` clause. You need to use different aliases for each table.

You need to add an index on the columns you want to join. The indexes will be discussed in Chapter 13, *Performance Tuning*. For now, you can execute this command to add an index:

```
mysql> ALTER TABLE employees ADD INDEX name(first_name,
last_name);
Query OK, 0 rows affected (1.95 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> SELECT
     emp1.*
 FROM
     employees emp1
 JOIN employees emp2
     ON emp1.first_name=emp2.first_name
     AND emp1.last_name=emp2.last_name
     AND emp1.gender=emp2.gender
     AND emp1.hire_date=emp2.hire_date
     AND emp1.emp_no!=emp2.emp_no
 ORDER BY
     first_name, last_name;
+--------+------------+------------+-----------+--------+---
---------+
| emp_no | birth_date | first_name | last_name | gender |
hire_date  |
+--------+------------+------------+-----------+--------+---
```

```
---------+
| 232772 | 1962-05-14 | Keung      | Heusch    | M      |
1986-06-01 |
| 493600 | 1964-01-26 | Keung      | Heusch    | M      |
1986-06-01 |
|  64089 | 1958-01-19 | Marit      | Kolvik    | F      |
1993-12-08 |
| 424486 | 1952-07-06 | Marit      | Kolvik    | F      |
1993-12-08 |
|  40965 | 1952-05-11 | Marsha     | Farrow    | M      |
1989-02-18 |
|  14641 | 1953-05-08 | Marsha     | Farrow    | M      |
1989-02-18 |
| 422332 | 1954-08-17 | Naftali    | Mawatari  | M      |
1985-09-14 |
| 427429 | 1962-11-06 | Naftali    | Mawatari  | M      |
1985-09-14 |
|  19454 | 1955-05-14 | Taisook    | Hutter    | F      |
1985-02-26 |
| 243627 | 1957-02-14 | Taisook    | Hutter    | F      |
1985-02-26 |
+--------+------------+------------+-----------+--------+---
---------+
10 rows in set (34.01 sec)
```

You have to mention `emp1.emp_no != emp2.emp_no` because the employees will have different `emp_no`. Otherwise, the same employee will appear.

Using SUB queries

A subquery is a `SELECT` statement within another statement.
Suppose you want to find the name of the employees who started as a `Senior Engineer` on `1986-06-26`.
You can get the `emp_no` from the `titles` table, and name from the `employees` table. You can also use `JOIN` to find out the results.

To get the `emp_no` from titles:

```
mysql> SELECT emp_no FROM titles WHERE title="Senior
Engineer" AND from_date="1986-06-26";
+--------+
| emp_no |
+--------+
|  10001 |
|  84305 |
| 228917 |
| 426700 |
| 458304 |
+--------+
5 rows in set (0.14 sec)
```

To find the name:

```
mysql> SELECT first_name, last_name FROM employees WHERE
emp_no IN (< output from preceding query>)

mysql> SELECT first_name, last_name FROM employees WHERE
emp_no IN (10001,84305,228917,426700,458304);
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| Georgi     | Facello   |
| Minghong   | Kalloufi  |
| Nechama    | Bennet    |
| Nagui      | Restivo   |
| Shuzo      | Kirkerud  |
+------------+-----------+
5 rows in set (0.00 sec
```

Other clauses such as `EXISTS` and `EQUAL` are also supported in MySQL. Refer to the reference manual, https://dev.mysql.com/doc/refman/8.0/en/subqueries.html, for more details:

```
mysql> SELECT
    first_name,
    last_name
FROM
    employees
WHERE
    emp_no
IN (SELECT emp_no FROM titles WHERE title="Senior Engineer"
AND from_date="1986-06-26");
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| Georgi     | Facello   |
| Minghong   | Kalloufi  |
| Nagui      | Restivo   |
| Nechama    | Bennet    |
| Shuzo      | Kirkerud  |
+------------+-----------+
5 rows in set (0.91 sec)
```

Find the employee making the maximum salary:

```
mysql> SELECT emp_no FROM salaries WHERE salary=(SELECT
MAX(salary) FROM salaries);
+--------+
| emp_no |
+--------+
|  43624 |
+--------+
1 row in set (1.54 sec)
```

`SELECT MAX(salary) FROM salaries` is the subquery that gives the maximum salary, to find the employee number corresponding to that salary, you can use that subquery in the `WHERE` clause.

Finding mismatched rows between tables

Suppose you want to find rows in a table that are not in other tables. You can achieve this in two ways. Using the `NOT IN` clause or using `OUTER JOIN`.

To find the matched rows, you can use normal `JOIN`, if you want to find mismatched rows, you can use `OUTER JOIN`. Normal `JOIN` means *A intersection B*. `OUTER JOIN` gives matching records of both *A* and *B* and also gives unmatched records of *A* with `NULL`. If you want the output of `A-B`, you can use the `WHERE <JOIN COLUMN IN B> IS NULL` clause.

To understand the usage of `OUTER JOIN`, create two `employee` tables and insert some values:

```
mysql> CREATE TABLE employees_list1 AS SELECT * FROM
employees WHERE first_name LIKE 'aa%';
Query OK, 444 rows affected (0.22 sec)
Records: 444  Duplicates: 0  Warnings: 0

mysql> CREATE TABLE employees_list2 AS SELECT * FROM
employees WHERE emp_no BETWEEN 400000 AND 500000 AND
gender='F';
Query OK, 39892 rows affected (0.59 sec)
Records: 39892  Duplicates: 0  Warnings: 0
```

You already know how to find the employees who exist in both lists:

```
mysql> SELECT * FROM employees_list1 WHERE emp_no IN (SELECT
emp_no FROM  employees_list2);
```

Or you can use `JOIN`:

```
mysql> SELECT l1.* FROM employees_list1 l1 JOIN
employees_list2 l2 ON l1.emp_no=l2.emp_no;
```

To find out the employees who exist in `employees_list1` but not in `employees_list2`:

```
mysql> SELECT * FROM employees_list1 WHERE emp_no NOT IN
(SELECT emp_no FROM  employees_list2);
```

Or you can use `OUTER JOIN`:

```
mysql> SELECT l1.* FROM employees_list1 l1 LEFT OUTER JOIN
employees_list2 l2 ON l1.emp_no=l2.emp_no WHERE l2.emp_no IS
NULL;
```

The outer join creates `NULL` columns of the second table in the join list for each unmatched row. If you use `RIGHT JOIN`, the first table will get `NULL` values for the unmatched rows.

You can also use `OUTER JOIN` to find matched rows. Instead of `WHERE l2.emp_no IS NULL`, give `WHERE emp_no IS NOT NULL`:

```
mysql> SELECT l1.* FROM employees_list1 l1 LEFT OUTER JOIN
employees_list2 l2 ON l1.emp_no=l2.emp_no WHERE l2.emp_no IS
NOT NULL;
```

Stored procedures

Suppose you need to execute a series of statements in MySQL, instead of sending all SQL statements every time, you can encapsulate all the statements in a single program and call it whenever required. A stored procedure is a set of SQL statements for which no return value is needed.

Apart from SQL statements, you can make use of variables to store results and do programmatical stuff inside a stored procedure. For example, you can write `IF`, `CASE` clauses, logical operations, and `WHILE` loops.

- Stored functions and procedures are also referred to as stored routines.
- For creating a stored procedure, you should have the `CREATE ROUTINE` privilege.
- Stored functions will have a return value.
- Stored procedures do not have a return value.
- All the code is written inside the `BEGIN and END` block.
- Stored functions can be called directly in a `SELECT` statement.
- Stored procedures can be called using the `CALL` statement.
- Since the statements inside stored routines should end with a delimiter (`;`), you have to change the delimiter for MySQL so that MySQL won't interpret the SQL statements inside a stored routine with normal statements. After the creation of the procedure, you can change the delimiter back to the default value.

How to do it...

For example, you want to add a new employee. You should update three tables, namely `employees`, `salaries`, and `titles`. Instead of executing three statements, you can develop a stored procedure and call it to create a new `employee`.

You have to pass the employee's `first_name`, `last_name`, `gender`, and `birth_date`, as well as the department the employee joins. You can pass those using input variables and you should get the employee number as output. The stored procedure does not return a value, but it can update a variable and you can use it.

Here is a simple example of a stored procedure to create a new employee and update the `salary` and `department` tables:

```
/* DROP the existing procedure if any with the same name
before creating */
DROP PROCEDURE IF EXISTS create_employee;
/* Change the delimiter to $$ */
DELIMITER $$
/* IN specifies the variables taken as arguments, INOUT
specifies the output variable*/
CREATE PROCEDURE create_employee (OUT new_emp_no INT, IN
first_name varchar(20), IN last_name varchar(20), IN gender
enum('M','F'), IN birth_date date, IN emp_dept_name
varchar(40), IN title varchar(50))
BEGIN
    /* Declare variables for emp_dept_no and salary */
        DECLARE emp_dept_no char(4);
        DECLARE salary int DEFAULT 60000;

    /* Select the maximum employee number into the variable
new_emp_no */
    SELECT max(emp_no) INTO new_emp_no FROM employees;
    /* Increment the new_emp_no */
    SET new_emp_no = new_emp_no + 1;

    /* INSERT the data into employees table */
        /* The function CURDATE() gives the current date) */
    INSERT INTO employees VALUES(new_emp_no, birth_date,
first_name, last_name, gender, CURDATE());

    /* Find out the dept_no for dept_name */
    SELECT emp_dept_name;
    SELECT dept_no INTO emp_dept_no FROM departments WHERE
dept_name=emp_dept_name;
    SELECT emp_dept_no;

    /* Insert into dept_emp */
    INSERT INTO dept_emp VALUES(new_emp_no, emp_dept_no,
CURDATE(), '9999-01-01');

    /* Insert into titles */
    INSERT INTO titles VALUES(new_emp_no, title, CURDATE(),
'9999-01-01');

    /* Find salary based on title */
    IF title = 'Staff'
        THEN SET salary = 100000;
    ELSEIF title = 'Senior Staff'
        THEN SET salary = 120000;
    END IF;

    /* Insert into salaries */
    INSERT INTO salaries VALUES(new_emp_no, salary,
CURDATE(), '9999-01-01');
END
$$
/* Change the delimiter back to ; */
DELIMITER ;
```

To create a stored procedure, you can:

- Paste it in the command-line client
- Save it in the file and import it in MySQL using `mysql -u <user> -p employees < stored_procedure.sql`
- Source the `mysql> SOURCE stored_procedure.sql` file

To use the stored procedure, grant the execute privilege to the `emp_read_only` user:

```
mysql> GRANT EXECUTE ON employees.* TO 'emp_read_only'@'%';
Query OK, 0 rows affected (0.05 sec)
```

Invoke the stored procedure using the `CALL stored_procedure(OUT variable, IN values)` statement and name of the routine.

Connect to MySQL using the `emp_read_only` account:

```
shell> mysql -u emp_read_only -pemp_pass employees -A
```

Pass the variable where you want to store the `@new_emp_no` output and also pass the required input values:

```
mysql> CALL create_employee(@new_emp_no, 'John', 'Smith',
'M', '1984-06-19', 'Research', 'Staff');
Query OK, 1 row affected (0.01 sec)
```

Select the value of `emp_no`, which is stored in the `@new_emp_no` variable:

```
mysql> SELECT @new_emp_no;
+-------------+
| @new_emp_no |
+-------------+
|      500000 |
+-------------+
1 row in set (0.00 sec)
```

Check that the rows are created in the `employees`, `salaries`, and `titles` tables:

```
mysql> SELECT * FROM employees WHERE emp_no=500000;
+--------+------------+------------+-----------+--------+---
---------+
| emp_no | birth_date | first_name | last_name | gender |
hire_date  |
+--------+------------+------------+-----------+--------+---
---------+
| 500000 | 1984-06-19 | John       | Smith     | M      |
2017-06-17 |
+--------+------------+------------+-----------+--------+---
---------+
1 row in set (0.00 sec)

mysql> SELECT * FROM salaries WHERE emp_no=500000;
+--------+--------+------------+------------+
| emp_no | salary | from_date  | to_date    |
+--------+--------+------------+------------+
| 500000 | 100000 | 2017-06-17 | 9999-01-01 |
+--------+--------+------------+------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM titles WHERE emp_no=500000;
+--------+-------+------------+------------+
| emp_no | title | from_date  | to_date    |
+--------+-------+------------+------------+
| 500000 | Staff | 2017-06-17 | 9999-01-01 |
+--------+-------+------------+------------+
1 row in set (0.00 sec)
```

You can see that, even though `emp_read_only` has no write access on the tables, it is able to write by calling the stored procedure. If the `SQL SECURITY` of the stored procedure is created as `INVOKER`, `emp_read_only` cannot modify the data. Note that if you are connecting using `localhost`, create the privileges for the `localhost` user.

To list all the stored procedures in a database, execute `SHOW PROCEDURE STATUS\G`. To check the definition of an existing stored routine, you can execute `SHOW CREATE PROCEDURE <procedure_name>\G`.

There's more...

Stored procedures are also used to enhance the security. The user needs the `EXECUTE` privilege on the stored procedure to execute it. By the definition of a stored routine:

- The `DEFINER` clause specifies the creator of the stored routine. If nothing is specified, the current user is taken.
- The `SQL SECURITY` clause specifies the execution context of the stored routine. It can be either `DEFINER` or `INVOKER`.

`DEFINER`: A user even with only the `EXECUTE` permission for routine can call and get the output of the stored routine, regardless of whether that user has permission on the underlying tables or not. It is enough if `DEFINER` has privileges.

`INVOKER`: The security context is switched to the user who invokes the stored routine. In this case, the invoker should have access to the underlying tables.

See also

Refer to the documentation for more examples and syntax, at https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html.

Functions

Just like stored procedures, you can create stored functions. The main difference is functions should have a return value and they can be called in SELECT. Usually, stored functions are created to simplify complex calculations.

How to do it...

Here is an example of how to write a function and how to call it. Suppose a banker wants to give a credit card based on income level, instead of exposing the actual salary, you can expose this function to find out the income level:

```
shell> vi function.sql;
DROP FUNCTION IF EXISTS get_sal_level;
DELIMITER $$
CREATE FUNCTION get_sal_level(emp int) RETURNS VARCHAR(10)
 DETERMINISTIC
BEGIN
 DECLARE sal_level varchar(10);
 DECLARE avg_sal FLOAT;

 SELECT AVG(salary) INTO avg_sal FROM salaries WHERE
emp_no=emp;

 IF avg_sal < 50000 THEN
 SET sal_level = 'BRONZE';
 ELSEIF (avg_sal >= 50000 AND avg_sal < 70000) THEN
 SET sal_level = 'SILVER';
 ELSEIF (avg_sal >= 70000 AND avg_sal < 90000) THEN
 SET sal_level = 'GOLD';
 ELSEIF (avg_sal >= 90000) THEN
 SET sal_level = 'PLATINUM';
 ELSE
 SET sal_level = 'NOT FOUND';
 END IF;
 RETURN (sal_level);
END
$$
DELIMITER ;
```

To create the function:

```
mysql> SOURCE function.sql;
Query OK, 0 rows affected (0.00 sec)
Query OK, 0 rows affected (0.01 sec)
```

```
You have to pass the employee number and the function
returns the income level.
mysql> SELECT get_sal_level(10002);
+----------------------+
| get_sal_level(10002) |
+----------------------+
| SILVER               |
+----------------------+
1 row in set (0.00 sec)

mysql> SELECT get_sal_level(10001);
+----------------------+
| get_sal_level(10001) |
+----------------------+
| GOLD                 |
+----------------------+
1 row in set (0.00 sec)

mysql> SELECT get_sal_level(1);
+------------------+
| get_sal_level(1) |
+------------------+
| NOT FOUND        |
+------------------+
1 row in set (0.00 sec)
```

To list all the stored functions in a database, execute `SHOW FUNCTION STATUS\G`. To check the definition of the existing stored function, you can execute `SHOW CREATE FUNCTION <function_name>\G`.

It is very important to give the `DETERMINISTIC` *keyword in the function creation. A routine is considered* `DETERMINISTIC` *if it always produces the same result for the same input parameters, and* `NOT DETERMINISTIC` *otherwise. If neither* `DETERMINISTIC` *nor* `NOT DETERMINISTIC` *is given in the routine definition, the default is* `NOT DETERMINISTIC`*. To declare that a function is deterministic, you must specify* `DETERMINISTIC` *explicitly.*

Declaring a `NON DETERMINISTIC` *routine as* `DETERMINISTIC` *might lead to unexpected results, by causing the optimizer to make incorrect execution plan choices. Declaring a* `DETERMINISTIC` *routine as* `NON DETERMINISTIC` *might diminish performance, by causing available optimizations not to be used.*

Inbuilt functions

MySQL provides numerous inbuilt functions. You have already used the `CURDATE()` function to get the current date.

You can use the functions in the `WHERE` clause:

```
mysql> SELECT * FROM employees WHERE hire_date = CURDATE();
```

- For example, the following function gives the date from exactly one week ago:

```
mysql> SELECT DATE_ADD(CURDATE(), INTERVAL -7 DAY) AS
'7 Days Ago';
```

- Add two strings:

```
mysql> SELECT CONCAT(first_name, ' ', last_name) FROM
employees LIMIT 1;
+------------------------------------+
| CONCAT(first_name, ' ', last_name) |
+------------------------------------+
| Aamer Anger                        |
+------------------------------------+
1 row in set (0.00 sec)
```

See also

Refer to the MySQL reference manual for a complete list of functions, at https://dev.mysql.com/doc/refman/8.0/en/func-op-summary-ref.html.

Triggers

A trigger is used to activate something before or after the trigger event. For example, you can have a trigger activate before each row that is inserted into a table or after each row that is updated.

Triggers are highly useful while altering a table without downtime (Refer to Chapter 10, *Table Maintenance*, in the *Alter table using online schema change tool* section) and also for auditing purposes. Suppose you want to find out the previous value of a row, you can write a trigger that saves those rows in another table before updating. The other table serves as an audit table that has the previous records.

The trigger action time can be `BEFORE` or `AFTER`, which indicates whether the trigger activates before or after each row to be modified.

Trigger events can be `INSERT`, `DELETE`, or `UPDATE`:

- `INSERT`: Whenever a new row gets inserted through `INSERT`, `REPLACE`, or `LOAD DATA`, the trigger gets activated
- `UPDATE`: Through the `UPDATE` statement
- `DELETE`: Through the `DELETE` or `REPLACE` statements

From MySQL 5.7, a table can have multiple triggers at the same time. For example, a table can have two `BEFORE INSERT` triggers. You have to specify which trigger should go first using `FOLLOWS` or `PRECEDES`.

How to do it...

For example, you want to round off the salary before inserting it into the `salaries` table. `NEW` refers to the new value that is being inserted:

```
shell> vi before_insert_trigger.sql
DROP TRIGGER IF EXISTS salary_round;
DELIMITER $$
CREATE TRIGGER salary_round BEFORE INSERT ON salaries
FOR EACH ROW
BEGIN
        SET NEW.salary=ROUND(NEW.salary);
END
$$
DELIMITER ;
```

Create the trigger by sourcing the file:

```
mysql> SOURCE before_insert_trigger.sql;
Query OK, 0 rows affected (0.06 sec)
Query OK, 0 rows affected (0.00 sec)
```

Test the trigger by inserting a floating number into the salary:

```
mysql> INSERT INTO salaries VALUES(10002, 100000.79,
CURDATE(), '9999-01-01');
Query OK, 1 row affected (0.04 sec)
```

You can see that the salary is rounded off:

```
mysql> SELECT * FROM salaries WHERE emp_no=10002 AND
from_date=CURDATE();
+--------+--------+------------+------------+
| emp_no | salary | from_date  | to_date    |
+--------+--------+------------+------------+
|  10002 | 100001 | 2017-06-18 | 9999-01-01 |
+--------+-------+-----------+-----------+
1 row in set (0.00 sec)
```

Similarly, you can create a `BEFORE UPDATE` trigger to round off the salary. Another example: you want to log which user has inserted into the `salaries` table. Create an `audit` table:

```
mysql> CREATE TABLE salary_audit (emp_no int, user
varchar(50), date_modified date);
```

Note that the following trigger precedes the `salary_round` trigger, which is specified by `PRECEDES salary_round`:

```
shell> vi before_insert_trigger.sql
DELIMITER $$
CREATE TRIGGER salary_audit
BEFORE INSERT
   ON salaries FOR EACH ROW PRECEDES salary_round
BEGIN
   INSERT INTO salary_audit VALUES(NEW.emp_no, USER(),
CURDATE());
END; $$
DELIMITER ;
```

Insert it into `salaries`:

```
mysql> INSERT INTO salaries VALUES(10003, 100000.79,
CURDATE(), '9999-01-01');
Query OK, 1 row affected (0.06 sec)
```

Find out who inserted the salary by querying the `salary_audit` table:

```
mysql> SELECT * FROM salary_audit WHERE emp_no=10003;
+--------+----------------+---------------+
| emp_no | user           | date_modified |
+--------+----------------+---------------+
|  10003 | root@localhost | 2017-06-18    |
+--------+----------------+---------------+
1 row in set (0.00 sec)
```

In case the `salary_audit` *table is dropped or is not available, all the inserts on the* `salaries` *table will be blocked. If you do not want to do auditing, you should drop the trigger first and then the table.*
Triggers can make overhead on the write speed based on the complexity of it.
To check all the triggers, execute `SHOW TRIGGERS\G`*.*
To check the definition of an existing trigger, execute `SHOW CREATE TRIGGER <trigger_name>`*.*

See also

Refer to the MySQL reference manual, at https://dev.mysql.com/doc/refman/8.0/en/trigger-syntax.html, for more details.

Views

View is a virtual table based on the result-set of an SQL statement. It will also have rows and columns just like a real table, but few restrictions, which will be discussed later. Views hide the SQL complexity and, more importantly, provide additional security.

How to do it...

Suppose you want to give access only to the emp_no and salary columns of the salaries table, and from_date is after 2002-01-01. For this, you can create a view with the SQL that gives the required result.

```
mysql> CREATE ALGORITHM=UNDEFINED
DEFINER=`root`@`localhost`
SQL SECURITY DEFINER VIEW salary_view
AS
SELECT emp_no, salary FROM salaries WHERE from_date > '2002-
01-01';
```

Now the salary_view view is created and you can query it just like any other table:

```
mysql> SELECT emp_no, AVG(salary) as avg FROM salary_view
GROUP BY emp_no ORDER BY avg DESC LIMIT 5;
```

You can see that the view has access to particular rows (that is, from_date > '2002-01-01') and not all of the rows. You can use the view to restrict user access to particular rows.

To list all views, execute:

```
mysql> SHOW FULL TABLES WHERE TABLE_TYPE LIKE 'VIEW';
```

To check the definition of the view, execute:

```
mysql> SHOW CREATE VIEW salary_view\G
```

You might have noticed the current_dept_emp and dept_emp_latest_date views, which are there as part of the employee database. You can explore the definition and find out their purpose.

Simple views that do not have sub-queries, `JOINS`, `GROUP BY` clauses, union, and so on, can be updated. `salary_view` is a simple view that can be updated or inserted if the underlying tables have a default value:

```
mysql> UPDATE salary_view SET salary=100000 WHERE
emp_no=10001;
Query OK, 1 row affected (0.01 sec)
Rows matched: 2 Changed: 1 Warnings: 0
```

```
mysql> INSERT INTO salary_view VALUES(10001,100001);
ERROR 1423 (HY000): Field of view 'employees.salary_view'
underlying table doesn't have a default value
```

If the table has a default value, you could have inserted a row even if it does not match the filter criteria in the view. To avoid that, and to insert rows that satisfy the view condition, you have to provide `WITH CHECK OPTION` in the definition.

The `VIEW` algorithms:

- `MERGE`: MySQL combines the input query and the view definition into a single query and then executes the combined query. The `MERGE` algorithm is allowed only on simple views.
- `TEMPTABLE`: MySQL stores the results in the temporary table and then it executes the input query against this temporary table.
- `UNDEFINED` (default): MySQL automatically chooses the `MERGE` or `TEMPTABLE` algorithm. MySQL prefers the `MERGE` algorithm to the `TEMPTABLE` algorithm because the `MERGE` algorithm is much more efficient.

Events

Just like a cron on a Linux server, MySQL has `EVENTS` to handle the scheduled tasks. MySQL uses a special thread called the event schedule thread to execute all scheduled events. By default, the event scheduler thread is not enabled (version < 8.0.3), to enable it, execute:

```
mysql> SET GLOBAL event_scheduler = ON;
```

How to do it...

Suppose you no longer need to keep salary audit records that are more than a month old, you can schedule an event that runs daily and deletes the records from the `salary_audit` table that are a month old.

```
mysql> DROP EVENT IF EXISTS purge_salary_audit;
DELIMITER $$
CREATE EVENT IF NOT EXISTS purge_salary_audit
ON SCHEDULE
  EVERY 1 WEEK
  STARTS CURRENT_DATE
    DO BEGIN
        DELETE FROM salary_audit WHERE date_modified <
DATE_ADD(CURDATE(), INTERVAL -7 day);
    END $$
DELIMITER ;
```

Once the event is created, it will automatically do the job of purging the salary audit records.

- To check the events, execute:

```
      mysql> SHOW EVENTS\G
      *************************** 1. row
      ***************************
                        Db: employees
                      Name: purge_salary_audit
                   Definer: root@localhost
                 Time zone: SYSTEM
                      Type: RECURRING
                Execute at: NULL
            Interval value: 1
            Interval field: MINUTE
                    Starts: 2017-06-18 00:00:00
                      Ends: NULL
                    Status: ENABLED
```

```
          Originator: 0
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: utf8mb4_0900_ai_ci
1 row in set (0.00 sec)
```

- To check the definition on the event, execute:

```
mysql> SHOW CREATE EVENT purge_salary_audit\G
```

- To disable/enable the event, execute the following:

```
mysql> ALTER EVENT purge_salary_audit DISABLE;
mysql> ALTER EVENT purge_salary_audit ENABLE;
```

Access control

All stored programs (procedures, functions, triggers, and events) and views have a `DEFINER`. If the `DEFINER` is not specified, the user who creates the object will be chosen as `DEFINER`.

Stored routines (procedures and functions) and views have an `SQL SECURITY` characteristic with a value of `DEFINER` or `INVOKER` to specify whether the object executes in the definer or invoker context. Triggers and events have no `SQL SECURITY` characteristic and always execute in the definer context. The server invokes these objects automatically as necessary, so there is no invoking user.

See also

There are lots of ways to schedule events, refer to https://dev.mysql.com/doc/refman/8.0/en/event-scheduler.html for more details.

Getting information about databases and tables

You might have already noticed an `information_schema` database in the list of databases. `information_schema` is a collection of views that consist of metadata about all the database objects. You can connect to `information_schema` and explore all the tables. The most widely-used tables are explained in this chapter. You either query the `information_schema` tables or use the `SHOW` command, which essentially does the same.

`INFORMATION_SCHEMA` queries are implemented as views over the `data dictionary` tables. There are two types of metadata in the `INFORMATION_SCHEMA` tables:

- **Static table metadata**: `TABLE_SCHEMA`, `TABLE_NAME`, `TABLE_TYPE`, and `ENGINE`. These statistics will be read directly from the `data dictionary`.
- **Dynamic table metadata**: `AUTO_INCREMENT`, `AVG_ROW_LENGTH`, and `DATA_FREE`. Dynamic metadata frequently changes (for example, the `AUTO_INCREMENT` value will advance after each `INSERT`). In many cases, the dynamic metadata will also incur some cost to accurately calculate on demand, and accuracy may not be beneficial for the typical query. Consider the case of the `DATA_FREE` statistic that shows the number of free bytes in a table—a cached value is usually sufficient.

In MySQL 8.0, the dynamic table metadata will default to being cached. This is configurable via the `information_schema_stats` setting (default cached), and can be changed to `SET @@GLOBAL.information_schema_stats='LATEST'` in order to always retrieve the dynamic information directly from the storage engine (at the cost of slightly higher query execution).

As an alternative, the user can also execute `ANALYZE TABLE` on the table, to update the cached dynamic statistics.

Most of the tables have the `TABLE_SCHEMA` column, which refers to the database name, and the `TABLE_NAME` column, which refers to the table name.

Refer to https://mysqlserverteam.com/mysql-8-0-improvements-to-information_schema/ for more details.

How to do it...

Check the list of all the tables:

```
mysql> USE INFORMATION_SCHEMA;
mysql> SHOW TABLES;
```

TABLES

The TABLES table contains all the information about the table, such as which database belongs to TABLE_SCHEMA, the number of rows (TABLE_ROWS), ENGINE, DATA_LENGTH, INDEX_LENGTH, and DATA_FREE:

```
mysql> DESC INFORMATION_SCHEMA.TABLES;
+-----------------+---------------------------------------
--------------------------+------+-----+-------------------
+-----------------------------+
| Field           | Type
| Null | Key | Default           | Extra
|
+-----------------+---------------------------------------
--------------------------+------+-----+-------------------
+-----------------------------+
| TABLE_CATALOG   | varchar(64)
| NO   |     | NULL              |
|
| TABLE_SCHEMA    | varchar(64)
| NO   |     | NULL              |
|
| TABLE_NAME      | varchar(64)
| NO   |     | NULL              |
|
| TABLE_TYPE      | enum('BASE TABLE','VIEW','SYSTEM VIEW')
| NO   |     | NULL              |
|
| ENGINE          | varchar(64)
| YES  |     | NULL              |
|
| VERSION         | int(2)
| YES  |     | NULL              |
|
| ROW_FORMAT      |
enum('Fixed','Dynamic','Compressed','Redundant','Compact','P
aged') | YES  |     | NULL              |
|
| TABLE_ROWS      | bigint(20) unsigned
| YES  |     | NULL              |
|
| AVG_ROW_LENGTH  | bigint(20) unsigned
| YES  |     | NULL              |
|
| DATA_LENGTH     | bigint(20) unsigned
| YES  |     | NULL              |
|
| MAX_DATA_LENGTH | bigint(20) unsigned
| YES  |     | NULL              |
|
| INDEX_LENGTH    | bigint(20) unsigned
| YES  |     | NULL              |
|
| DATA_FREE       | bigint(20) unsigned
| YES  |     | NULL              |
|
| AUTO_INCREMENT  | bigint(20) unsigned
| YES  |     | NULL              |
|
| CREATE_TIME     | timestamp
| NO   |     | CURRENT_TIMESTAMP | on update
CURRENT_TIMESTAMP |
| UPDATE_TIME     | timestamp
| YES  |     | NULL              |
|
| CHECK_TIME      | timestamp
| YES  |     | NULL              |
|
| TABLE_COLLATION | varchar(64)
| YES  |     | NULL              |
|
| CHECKSUM        | bigint(20) unsigned
| YES  |     | NULL              |
|
| CREATE_OPTIONS  | varchar(256)
| YES  |     | NULL              |
|
| TABLE_COMMENT   | varchar(256)
| YES  |     | NULL              |
|
+-----------------+---------------------------------------
--------------------------+------+-----+-------------------
+---------------------------+
21 rows in set (0.00 sec)
```

For example, you want to know `DATA_LENGTH`, `INDEX_LENGTH`, and `DATE_FREE` inside the `employees` database:

```
mysql> SELECT SUM(DATA_LENGTH)/1024/1024 AS DATA_SIZE_MB,
SUM(INDEX_LENGTH)/1024/1024 AS INDEX_SIZE_MB,
SUM(DATA_FREE)/1024/1024 AS DATA_FREE_MB FROM
INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA='employees';
+--------------+---------------+--------------+
| DATA_SIZE_MB | INDEX_SIZE_MB | DATA_FREE_MB |
+--------------+---------------+--------------+
|  17.39062500 |   14.62500000 |  11.00000000 |
+--------------+---------------+--------------+
1 row in set (0.01 sec)
```

COLUMNS

This table lists all the columns and its definition for each table:

```
mysql> SELECT * FROM COLUMNS WHERE TABLE_NAME='employees'\G
```

FILES

You have already seen that MySQL stores the `InnoDB` data in the `.ibd` files inside a directory (with the same name as the database name) in the `data directory`. To get more information about the files, you can query the `FILES` table:

```
mysql> SELECT * FROM FILES WHERE FILE_NAME LIKE
'./employees/employees.ibd'\G
~~~
EXTENT_SIZE: 1048576
AUTOEXTEND_SIZE: 4194304
DATA_FREE: 13631488
~~~
```

You should be keen at `DATA_FREE`, which represents the unallocated segments plus the data that is free inside the segments due to fragmentation. When you rebuild the table, you can free up bytes shown in `DATA_FREE`.

INNODB_SYS_TABLESPACES

The size of the file is also available in the `INNODB_TABLESPACES` table:

```
mysql> SELECT * FROM INNODB_TABLESPACES WHERE
NAME='employees/employees'\G
*************************** 1. row
***************************
 SPACE: 118
 NAME: employees/employees
 FLAG: 16417
 ROW_FORMAT: Dynamic
 PAGE_SIZE: 16384
 ZIP_PAGE_SIZE: 0
 SPACE_TYPE: Single
 FS_BLOCK_SIZE: 4096
 FILE_SIZE: 32505856
ALLOCATED_SIZE: 32509952
1 row in set (0.00 sec)
```

You can verify the same in the filesystem:

```
shell> sudo ls -ltr /var/lib/mysql/employees/employees.ibd
-rw-r----- 1 mysql mysql 32505856 Jun 20 16:50
/var/lib/mysql/employees/employees.ibd
```

INNODB_TABLESTATS

The size of the index and approximate number of rows is available in the `INNODB_TABLESTATS` table:

```
mysql> SELECT * FROM INNODB_TABLESTATS WHERE
NAME='employees/employees'\G
*************************** 1. row
***************************
 TABLE_ID: 128
 NAME: employees/employees
 STATS_INITIALIZED: Initialized
 NUM_ROWS: 299468
 CLUST_INDEX_SIZE: 1057
 OTHER_INDEX_SIZE: 545
 MODIFIED_COUNTER: 0
 AUTOINC: 0
 REF_COUNT: 1
1 row in set (0.00 sec)
```

PROCESSLIST

One of the most used views is the process list. It lists all the queries running on the server:

```
mysql> SELECT * FROM PROCESSLIST\G
*************************** 1. row
***************************
     ID: 85
   USER: event_scheduler
   HOST: localhost
     DB: NULL
COMMAND: Daemon
   TIME: 44
  STATE: Waiting for next activation
   INFO: NULL
*************************** 2. row
***************************
     ID: 26231
   USER: root
   HOST: localhost
     DB: information_schema
COMMAND: Query
   TIME: 0
  STATE: executing
   INFO: SELECT * FROM PROCESSLIST
2 rows in set (0.00 sec
```

Or you can execute `SHOW PROCESSLIST;` to get the same output.

Other tables: `ROUTINES` contains the definition of functions and stored routines. `TRIGGERS` contains the definition of triggers. `VIEWS` contains the definition of views.

See also

To learn about the improvements in `INFORMATION_SCHEMA`, refer to http://mysqlserverteam.com/mysql-8-0-improvements-to-information_schema/.

Using MySQL (Advanced)

In this chapter, we will cover the following recipes:

- Using JSON
- Common table expressions (CTE)
- Generated columns
- Window functions

Introduction

In this chapter, you will learn about the newly introduced features of MySQL.

Using JSON

As you have seen in the previous chapter, to store data in MySQL, you have to define the database and table structure (schema), which is a major limitation. To cope with that, from MySQL 5.7, MySQL supports the **JavaScript Object Notation** (**JSON**) datatype. Earlier there was no separate datatype and it was stored as a string. The new JSON datatype provides automatic validation of JSON documents and optimized storage format.

JSON documents are stored in binary format, which enables the following:

- Quick-read access to document elements
- No need for the value to be parsed from a text representation when the server reads the JSON again
- Looking up subobjects or nested values directly by key or array index without reading all values before or after them in the document

How to do it...

Suppose you want to store more details about your employees; you can save them using JSON:

```
CREATE TABLE emp_details(
  emp_no int primary key,
  details json
);
```

Insert JSON

```
INSERT INTO emp_details(emp_no, details)
VALUES ('1',
'{ "location": "IN", "phone": "+11800000000", "email":
"abc@example.com", "address": { "line1": "abc", "line2":
"xyz street", "city": "Bangalore", "pin": "560103"} }'
);
```

Retrieve JSON

You can retrieve the fields of the JSON column using the `->` and `->>` operators:

```
mysql> SELECT emp_no, details->'$.address.pin' pin FROM
emp_details;
+--------+----------+
| emp_no | pin      |
+--------+----------+
| 1      | "560103" |
+--------+----------+
1 row in set (0.00 sec)
```

To retrieve data without quotes, use the `->>` operator:

```
mysql> SELECT emp_no, details->>'$.address.pin' pin FROM
emp_details;
+--------+--------+
| emp_no | pin    |
+--------+--------+
| 1      | 560103 |
+--------+--------+
1 row in set (0.00 sec)
```

JSON functions

MySQL provides many functions to deal with JSON data. Let's look into the most used ones.

Pretty view

To display JSON values in pretty format, use the `JSON_PRETTY()` function:

```
mysql> SELECT emp_no, JSON_PRETTY(details) FROM emp_details
\G
*************************** 1. row
***************************
 emp_no: 1
JSON_PRETTY(details): {
 "email": "abc@example.com",
 "phone": "+11800000000",
 "address": {
 "pin": "560103",
 "city": "Bangalore",
 "line1": "abc",
 "line2": "xyz street"
 },
 "location": "IN"
}
1 row in set (0.00 sec)
```

Without pretty:

```
mysql> SELECT emp_no, details FROM emp_details \G
*************************** 1. row
***************************
 emp_no: 1
details: {"email": "abc@example.com", "phone":
"+11800000000", "address": {"pin": "560100", "city":
"Bangalore", "line1": "abc", "line2": "xyz street"},
"location": "IN"}
1 row in set (0.00 sec)
```

Searching

You can reference a JSON column using the `col->>path` operator in the `WHERE` clause:

```
mysql> SELECT emp_no FROM emp_details WHERE details-
>>'$.address.pin'="560103";
+--------+
| emp_no |
+--------+
| 1      |
+--------+
1 row in set (0.00 sec)
```

You can also use the `JSON_CONTAINS` function to search for data. It returns `1` if the data is found and `0` otherwise:

```
mysql> SELECT JSON_CONTAINS(details->>'$.address.pin',
"560103") FROM emp_details;
+------------------------------------------------------+
| JSON_CONTAINS(details->>'$.address.pin', "560103") |
+------------------------------------------------------+
| 1                                                    |
+------------------------------------------------------+
1 row in set (0.00 sec)
```

How to search for a key? Suppose you want to check whether `address.line1` exists or not:

```
mysql> SELECT JSON_CONTAINS_PATH(details, 'one',
"$.address.line1") FROM emp_details;
+------------------------------------------------------------
---------------+
| JSON_CONTAINS_PATH(details, 'one', "$.address.line1")
|
+------------------------------------------------------------
---------------+
| 1
```

```
|
+------------------------------------------------------------
---------------+
1 row in set (0.01 sec)
```

Here, `one` indicates at least one key should exists. Suppose you want to check whether `address.line1` or `address.line2` exists:

```
mysql> SELECT JSON_CONTAINS_PATH(details, 'one',
"$.address.line1", "$.address.line5") FROM emp_details;
+------------------------------------------------------------
---------------+
| JSON_CONTAINS_PATH(details, 'one', "$.address.line1",
"$.address.line2") |
+------------------------------------------------------------
---------------+
| 1
|
+------------------------------------------------------------
---------------+
```

If you want to check whether both `address.line1` and `address.line5` exist, you can use `and` instead of `one`:

```
mysql> SELECT JSON_CONTAINS_PATH(details, 'all',
"$.address.line1", "$.address.line5") FROM emp_details;
+------------------------------------------------------------
---------------+
| JSON_CONTAINS_PATH(details, 'all', "$.address.line1",
"$.address.line5") |
+------------------------------------------------------------
---------------+
| 0
|
+------------------------------------------------------------
---------------+
1 row in set (0.00 sec)
```

Modifying

You can modify the data using three different functions: `JSON_SET()`, `JSON_INSERT()`, `JSON_REPLACE()`. Before MySQL 8, we needed a full update of the entire column, which is not the optimal way:

- `JSON_SET`: Replaces existing values and adds non-existing values.
  Suppose you want to replace the pin code of the employee and also add details of a nickname:

```
mysql> UPDATE
    emp_details
SET
    details = JSON_SET(details, "$.address.pin",
"560100", "$.nickname", "kai")
WHERE
    emp_no = 1;
Query OK, 1 row affected (0.03 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

- `JSON_INSERT()`: Inserts values without replacing existing values

  Suppose you want to add a new column without updating the existing values; you can use `JSON_INSERT()`:

```
mysql> UPDATE emp_details SET
details=JSON_INSERT(details, "$.address.pin", "560132",
"$.address.line4", "A Wing") WHERE emp_no = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

  In this case, `pin` won't be updated; only a new `address.line4` field will be added.

- `JSON_REPLACE()`: Replaces *only* existing values

  Suppose you want to just replace the fields without adding new fields:

```
mysql> UPDATE emp_details SET
details=JSON_REPLACE(details, "$.address.pin",
"560132", "$.address.line5", "Landmark") WHERE
emp_no = 1;
Query OK, 1 row affected (0.04 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

In this case, `line5` won't be added. Only `pin` will be updated.

Removing

`JSON_REMOVE` removes data from a JSON document.

Suppose you no longer need `line5` in the address:

```
mysql> UPDATE emp_details SET details=JSON_REMOVE(details,
"$.address.line5") WHERE emp_no = 1;
Query OK, 1 row affected (0.04 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

Other functions

Some of the other functions are as follows:

- `JSON_KEYS()`: Gets all the keys in a JSON document:

```
mysql> SELECT JSON_KEYS(details) FROM emp_details WHERE
emp_no = 1;
*************************** 1. row
***************************
JSON_KEYS(details): ["email", "phone", "address",
"nickname", "locatation"]
```

- `JSON_LENGTH()`: Gives the number of elements in a JSON document:

```
mysql> SELECT JSON_LENGTH(details) FROM emp_details
WHERE emp_no = 1;
*************************** 1. row
***************************
JSON_LENGTH(details): 5
```

See also

You can check out the complete list of functions at https://dev.mysql.com/doc/refman/8.0/en/json-function-reference.html.

Common table expressions (CTE)

MySQL 8 supports common table expressions, both non-recursive and recursive.

Common table expressions enable the use of named temporary result sets, implemented by permitting a `WITH` clause preceding `SELECT` statements and certain other statements.

Why do you need CTEs?
It is not possible to refer to a derived table twice in the same query. So the derived tables are evaluated twice or as many times as referred, which indicates a serious performance problem. Using CTE, the subquery is evaluated only once.

How to do it...

Recursive and non-recursive CTE will be looked into in the following sections.

Non-recursive CTE

A Common table expression (**CTE**) is just like a derived table, but its declaration is put before the query block instead of in `FROM` clause.

Derived Table

```
SELECT... FROM (subquery) AS derived, t1 ...
```

CTE

```
SELECT... WITH derived AS (subquery) SELECT ... FROM
derived, t1 ...
```

A CTE may precede `SELECT`/`UPDATE`/`DELETE`, including subqueries `WITH` derived `AS` (subquery), for example:

```
DELETE FROM t1 WHERE t1.a IN (SELECT b FROM derived);
```

Suppose you want to find out the percentage increase in the salary of each year with respect to the previous year. Without CTE, you need to write two subqueries, and they are essentially the same. MySQL is not smart enough to detect that and the subqueries are executed twice:

```
mysql> SELECT
    q1.year,
    q2.year AS next_year,
    q1.sum,
    q2.sum AS next_sum,
    100*(q2.sum-q1.sum)/q1.sum AS pct
FROM
    (SELECT year(from_date) as year, sum(salary) as sum FROM
salaries GROUP BY year) AS q1,             (SELECT
year(from_date) as year, sum(salary) as sum FROM salaries
GROUP BY year) AS q2
WHERE q1.year = q2.year-1;
+------+-----------+-------------+-------------+----------+
| year | next_year | sum         | next_sum    | pct      |
+------+-----------+-------------+-------------+----------+
| 1985 |      1986 | 972864875   | 2052895941  | 111.0155 |
| 1986 |      1987 | 2052895941  | 3156881054  | 53.7770  |
| 1987 |      1988 | 3156881054  | 4295598688  | 36.0710  |
| 1988 |      1989 | 4295598688  | 5454260439  | 26.9732  |
| 1989 |      1990 | 5454260439  | 6626146391  | 21.4857  |
| 1990 |      1991 | 6626146391  | 7798804412  | 17.6974  |
| 1991 |      1992 | 7798804412  | 9027872610  | 15.7597  |
| 1992 |      1993 | 9027872610  | 10215059054 | 13.1502  |
| 1993 |      1994 | 10215059054 | 11429450113 | 11.8882  |
| 1994 |      1995 | 11429450113 | 12638817464 | 10.5812  |
| 1995 |      1996 | 12638817464 | 13888587737 | 9.8883   |
| 1996 |      1997 | 13888587737 | 15056011781 | 8.4056   |
| 1997 |      1998 | 15056011781 | 16220495471 | 7.7343   |
| 1998 |      1999 | 16220495471 | 17360258862 | 7.0267   |
| 1999 |      2000 | 17360258862 | 17535667603 | 1.0104   |
| 2000 |      2001 | 17535667603 | 17507737308 | -0.1593  |
| 2001 |      2002 | 17507737308 | 10243358658 | -41.4924 |
+------+-----------+-------------+-------------+----------+
17 rows in set (3.22 sec)
```

With non-recursive CTE, the derived query is executed only once and reused:

```
mysql>
WITH CTE AS
    (SELECT year(from_date) AS year, SUM(salary) AS sum FROM
salaries GROUP BY year)
SELECT
    q1.year, q2.year as next_year, q1.sum, q2.sum as
next_sum, 100*(q2.sum-q1.sum)/q1.sum as pct FROM
    CTE AS q1,
    CTE AS q2
WHERE
    q1.year = q2.year-1;
+------+-----------+-------------+-------------+----------+
| year | next_year | sum         | next_sum    | pct      |
+------+-----------+-------------+-------------+----------+
| 1985 |      1986 | 972864875   | 2052895941  | 111.0155 |
| 1986 |      1987 | 2052895941  | 3156881054  | 53.7770  |
| 1987 |      1988 | 3156881054  | 4295598688  | 36.0710  |
| 1988 |      1989 | 4295598688  | 5454260439  | 26.9732  |
| 1989 |      1990 | 5454260439  | 6626146391  | 21.4857  |
| 1990 |      1991 | 6626146391  | 7798804412  | 17.6974  |
| 1991 |      1992 | 7798804412  | 9027872610  | 15.7597  |
| 1992 |      1993 | 9027872610  | 10215059054 | 13.1502  |
| 1993 |      1994 | 10215059054 | 11429450113 | 11.8882  |
| 1994 |      1995 | 11429450113 | 12638817464 | 10.5812  |
| 1995 |      1996 | 12638817464 | 13888587737 | 9.8883   |
| 1996 |      1997 | 13888587737 | 15056011781 | 8.4056   |
```

```
| 1997 |      1998 | 15056011781 | 16220495471 | 7.7343   |
| 1998 |      1999 | 16220495471 | 17360258862 | 7.0267   |
| 1999 |      2000 | 17360258862 | 17535667603 | 1.0104   |
| 2000 |      2001 | 17535667603 | 17507737308 | -0.1593  |
| 2001 |      2002 | 17507737308 | 10243358658 | -41.4924 |
+------+----------+-------------+-------------+----------+
17 rows in set (1.63 sec)
```

You may notice that with CTE, the results are the same and query time improves by 50%; the readability is good and can be referenced multiple times.

Derived queries cannot refer to other derived queries:

```
SELECT ...
 FROM (SELECT ... FROM ...) AS d1, (SELECT ... FROM d1 ...)
AS d2 ...
ERROR: 1146 (42S02): Table ‘db.d1’ doesn’t exist
```

CTEs can refer to other CTEs:

```
WITH d1 AS (SELECT ... FROM ...), d2 AS (SELECT ... FROM d1
...)
SELECT
 FROM d1, d2 ...
```

Recursive CTE

A recursive CTE is a CTE with a subquery that refers to its own name. The `WITH` clause must begin with `WITH RECURSIVE`. The recursive CTE subquery has two parts, seed query and recursive query, separated by `UNION [ALL]` or `UNION DISTINCT`.

Seed `SELECT` is executed once to create the initial data subset; recursive `SELECT` is repeatedly executed to return subsets of data until the complete result set is obtained. Recursion stops when an iteration does not generate any new rows. This is useful to dig into hierarchies (parent/child or part/subpart):

```
WITH RECURSIVE cte AS
(SELECT ... FROM table_name /* seed SELECT */
UNION ALL
SELECT ... FROM cte, table_name) /* "recursive" SELECT */
SELECT ... FROM cte;
```

Suppose you want to print all numbers from `1` to `5`:

```
mysql> WITH RECURSIVE cte (n) AS
( SELECT 1 /* seed query */
  UNION ALL
  SELECT n + 1 FROM cte WHERE n < 5 /* recursive query */
)
SELECT * FROM cte;
+---+
| n |
+---+
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
+---+
5 rows in set (0.00 sec)
```

At each iteration, `SELECT` produces a row with a new value one more than the value of `n` from the previous row set. The first iteration operates on the initial row set (1) and produces *1+1=2*; the second iteration operates on the first iteration's row set (2) and produces *2+1=3*; and so on and so forth. This continues until the recursion ends, when `n` is no longer less than `5`.

Suppose you want to do hierarchical data traversal to produce an organizational chart with the management chain for each employee (that is, the path from CEO to employee). Use a recursive CTE!

Create a test table with `manager_id`:

```
mysql> CREATE TABLE employees_mgr (
 id INT PRIMARY KEY NOT NULL,
 name VARCHAR(100) NOT NULL,
 manager_id INT NULL,
 INDEX (manager_id),
FOREIGN KEY (manager_id) REFERENCES employees_mgr (id)
);
```

Insert sample data:

```
mysql> INSERT INTO employees_mgr VALUES
(333, "Yasmina", NULL), # Yasmina is the CEO (manager_id is
NULL)
(198, "John", 333), # John has ID 198 and reports to 333
(Yasmina)
(692, "Tarek", 333),
(29, "Pedro", 198),
(4610, "Sarah", 29),
(72, "Pierre", 29),
(123, "Adil", 692);
```

Execute the recursive CTE:

```
mysql> WITH RECURSIVE employee_paths (id, name, path) AS
(
 SELECT id, name, CAST(id AS CHAR(200))
 FROM employees_mgr
 WHERE manager_id IS NULL
 UNION ALL
 SELECT e.id, e.name, CONCAT(ep.path, ',', e.id)
 FROM employee_paths AS ep JOIN employees_mgr AS e
 ON ep.id = e.manager_id
)
SELECT * FROM employee_paths ORDER BY path;
```

It produces the following results:

```
+------+---------+-----------------+
| id   | name    | path            |
+------+---------+-----------------+
| 333  | Yasmina | 333             |
| 198  | John    | 333,198         |
| 29   | Pedro   | 333,198,29      |
| 4610 | Sarah   | 333,198,29,4610 |
| 72   | Pierre  | 333,198,29,72   |
| 692  | Tarek   | 333,692         |
| 123  | Adil    | 333,692,123     |
+------+---------+-----------------+
7 rows in set (0.00 sec)
```

`WITH RECURSIVE employee_paths (id, name, path) AS` is the name of the CTE and the columns are (`id`, `name`, `path`).

`SELECT id, name, CAST(id AS CHAR(200)) FROM employees_mgr WHERE manager_id IS NULL` is the seed query where the CEO is selected (CEO does not have a manager).

`SELECT e.id, e.name, CONCAT(ep.path, ',', e.id) FROM employee_paths AS ep JOIN employees_mgr AS e ON ep.id = e.manager_id` is the recursive query.

Each row produced by the recursive query finds all employees who report directly to an employee produced by a previous row. For every such employee, the row includes the employee ID, name, and employee management chain. The chain is the manager's chain with the employee ID added at the end.

Generated columns

The generated columns are also known as virtual or computed columns. The values of a generated column are computed from an expression included in the column definition. There are two types of generated columns:

- **Virtual**: The column will be calculated on the fly when a record is read from a table
- **Stored**: The column will be calculated when a new record is written in the table and will be stored in the table as a regular column

Virtual generated columns are more useful than stored generated columns because a virtual column does not take any storage space. You can simulate the behavior of stored generated columns using triggers.

How to do it...

Suppose your application is using `full_name` as `concat('first_name', ' ', 'last_name')` while retrieving the data from the `employees` table; instead of using the expression, you can use a virtual column, which calculates `full_name` on the fly. You can add another column followed by the expression:

```
mysql> CREATE TABLE `employees` (
  `emp_no` int(11) NOT NULL,
  `birth_date` date NOT NULL,
  `first_name` varchar(14) NOT NULL,
  `last_name` varchar(16) NOT NULL,
  `gender` enum('M','F') NOT NULL,
  `hire_date` date NOT NULL,
  `full_name` VARCHAR(30) AS (CONCAT(first_name,'
',last_name)),
  PRIMARY KEY (`emp_no`),
  KEY `name` (`first_name`,`last_name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Note that you should modify the insert statement according to the virtual column. You have an option to use the full insert like this:

```
mysql> INSERT INTO employees (emp_no, birth_date,
first_name, last_name, gender, hire_date) VALUES (123456,
'1987-10-02', 'ABC' , 'XYZ', 'F', '2008-07-28');
```

If you want to include `full_name` in the `INSERT` statement, you can specify it as `DEFAULT` only. All other values throw an `ERROR 3105 (HY000):` error. The value specified for the generated column, `full_name`, in the `employees` table is not allowed:

```
mysql> INSERT INTO employees (emp_no, birth_date,
first_name, last_name, gender, hire_date, full_name) VALUES
```

```
(123456, '1987-10-02', 'ABC' , 'XYZ', 'F', '2008-07-28',
DEFAULT);
```

You can directly select `full_name` from the `employees` table:

```
mysql> SELECT * FROM employees WHERE emp_no=123456;
+--------+------------+------------+-----------+--------+---
---------+-----------+
| emp_no | birth_date | first_name | last_name | gender |
hire_date  | full_name |
+--------+------------+------------+-----------+--------+---
---------+-----------+
| 123456 | 1987-10-02 | ABC        | XYZ       | F      |
2017-11-23 | ABC XYZ   |
+--------+------------+------------+-----------+--------+---
---------+-----------+
1 row in set (0.00 sec)
```

If you have already created the table and wish to add new generated column, execute the ALTER TABLE statement which will be covered in detail in *Chapter 10 - Table Maintenance*

Example:

```
mysql> ALTER TABLE employees ADD hire_date_year YEAR AS
(YEAR(hire_date)) VIRTUAL;
```

Refer to https://dev.mysql.com/doc/refman/8.0/en/create-table-generated-columns.html to know more about generated columns. You will understand the other uses of virtual columns in Chapter 13, *Performance Tuning*, in the *Adding indexes* and *Index for JSON using generated columns* sections.

Window functions

By using window functions, for each row from a query, you can perform a calculation using rows related to that row. This is accomplished by using the `OVER` and `WINDOW` clauses.

Here is the list of calculations that you can do:

- `CUME_DIST()`: Cumulative distribution value
- `DENSE_RANK()`: Rank of the current row within its partition, without gaps
- `FIRST_VALUE()`: The value of the argument from the first row of the window frame
- `LAG()`: The argument value from the row lagging the current row within partition
- `LAST_VALUE()`: Value of argument from the first row of window frame
- `LEAD()`: Value of argument from row leading current row within partition
- `NTH_VALUE()`: Argument value from *n*-th row of window frame
- `NTILE()`: Bucket number of the current row within its partition
- `PERCENT_RANK()`: Percentage rank value
- `RANK()`: Rank of the current row within its partition, with gaps
- `ROW_NUMBER()`: Number of the current row within its partition

How to do it...

Window functions can be used in various ways. Let's get to grips with each one of them in the following sections. For these examples to work, you need to add hire_date_year virtual column

```
mysql> ALTER TABLE employees ADD hire_date_year YEAR AS
(YEAR(hire_date)) VIRTUAL;
```

Row number

You can get the row number for each row to rank the results:

```
mysql> SELECT CONCAT(first_name, " ", last_name) AS
full_name, salary, ROW_NUMBER() OVER(ORDER BY salary DESC)
AS 'Rank'  FROM employees JOIN salaries ON
salaries.emp_no=employees.emp_no LIMIT 10;
+-------------------+--------+------+
| full_name         | salary | Rank |
+-------------------+--------+------+
| Tokuyasu Pesch    | 158220 |    1 |
| Tokuyasu Pesch    | 157821 |    2 |
| Honesty Mukaidono | 156286 |    3 |
| Xiahua Whitcomb   | 155709 |    4 |
| Sanjai Luders     | 155513 |    5 |
| Tsutomu Alameldin | 155377 |    6 |
| Tsutomu Alameldin | 155190 |    7 |
| Tsutomu Alameldin | 154888 |    8 |
| Tsutomu Alameldin | 154885 |    9 |
| Willard Baca      | 154459 |   10 |
+-------------------+--------+------+
10 rows in set (6.24 sec)
```

Partition results

You can partition the results in the `OVER` clause. Suppose you want to find out the salary rank for each year; it can be done as follows:

```
mysql> SELECT hire_date_year, salary, ROW_NUMBER()
OVER(PARTITION BY hire_date_year ORDER BY salary DESC) AS
'Rank' FROM employees JOIN salaries ON
salaries.emp_no=employees.emp_no ORDER BY salary DESC LIMIT
10;
+----------------+--------+------+
| hire_date_year | salary | Rank |
+----------------+--------+------+
|           1985 | 158220 |    1 |
|           1985 | 157821 |    2 |
|           1986 | 156286 |    1 |
|           1985 | 155709 |    3 |
|           1987 | 155513 |    1 |
|           1985 | 155377 |    4 |
|           1985 | 155190 |    5 |
|           1985 | 154888 |    6 |
|           1985 | 154885 |    7 |
|           1985 | 154459 |    8 |
+----------------+--------+------+
10 rows in set (8.04 sec)
```

You can notice that the rank changed for the years `1986` and `1987` but the ranking for `1985` continued.

Named windows

You can name a window and use it as many times as you need rather than redefining it every time:

```
mysql> SELECT hire_date_year, salary, RANK() OVER w AS
'Rank' FROM employees join salaries ON
salaries.emp_no=employees.emp_no WINDOW w AS (PARTITION BY
hire_date_year ORDER BY salary DESC) ORDER BY salary DESC
LIMIT 10;
+----------------+--------+------+
| hire_date_year | salary | Rank |
+----------------+--------+------+
|           1985 | 158220 |    1 |
|           1985 | 157821 |    2 |
|           1986 | 156286 |    1 |
|           1985 | 155709 |    3 |
|           1987 | 155513 |    1 |
|           1985 | 155377 |    4 |
|           1985 | 155190 |    5 |
|           1985 | 154888 |    6 |
|           1985 | 154885 |    7 |
|           1985 | 154459 |    8 |
+----------------+--------+------+
10 rows in set (8.52 sec)
```

First, last, and nth values

You can select the first, last, and *n*th values in the window results. If the row does not exist, a `NULL` value is returned.

Suppose you want to find the first, last, and third values from the window:

```
mysql> SELECT hire_date_year, salary, RANK() OVER w AS
'Rank',
FIRST_VALUE(salary) OVER w AS 'first',
NTH_VALUE(salary, 3) OVER w AS 'third',
LAST_VALUE(salary) OVER w AS 'last'
FROM employees join salaries ON
salaries.emp_no=employees.emp_no
WINDOW w AS (PARTITION BY hire_date_year ORDER BY salary
DESC)
ORDER BY salary DESC LIMIT 10;
+----------------+--------+------+--------+--------+--------
+
| hire_date_year | salary | Rank | first  | third  | last
|
+----------------+--------+------+--------+--------+--------
+
|           1985 | 158220 |    1 | 158220 |   NULL | 158220
|
|           1985 | 157821 |    2 | 158220 |   NULL | 157821
|
|           1986 | 156286 |    1 | 156286 |   NULL | 156286
|
|           1985 | 155709 |    3 | 158220 | 155709 | 155709
|
|           1987 | 155513 |    1 | 155513 |   NULL | 155513
|
|           1985 | 155377 |    4 | 158220 | 155709 | 155377
|
|           1985 | 155190 |    5 | 158220 | 155709 | 155190
|
|           1985 | 154888 |    6 | 158220 | 155709 | 154888
|
|            1985 | 154885 |    7 | 158220 | 155709 | 154885
|
|            1985 | 154459 |    8 | 158220 | 155709 | 154459
|
+----------------+--------+------+--------+--------+--------
+
10 rows in set (12.88 sec)
```

To know more about the other use cases of window functions, refer to https://mysqlserverteam.com/mysql-8-0-2-introducing-window-functions and https://dev.mysql.com/doc/refman/8.0/en/window-function-descriptions.html#function_row-number.

Configuring MySQL

In this chapter, we will cover the following recipes:

- Using config file
- Using global and session variables
- Using parameters with startup script
- Configuring the parameters
- Changing the data directory

Introduction

MySQL has two types of parameters:

- **Static**, which takes effect after restarting MySQL server
- **Dynamic**, which can be changed on the fly without restarting MySQL server

Variables can be set through the following:

- **Config file**: MySQL has a configuration file in which we can specify the location of data, the memory that MySQL can use, and various other parameters.
- **Startup script**: You can directly pass the parameters to the `mysqld` process. It remains in effect only for that invocation of the server.
- **Using SET command** (only dynamic variables): This will last until the server restarts. You also need to set the variable in the config file to make the change persistent across restarts. Another way to make changes persistent is by preceding the variable name by the `PERSIST` keyword or `@@persist`.

Using config file

The default config file is `/etc/my.cnf` (on Red Hat and CentOS systems) and `/etc/mysql/my.cnf` (Debian systems). Open the file in your favorite editor and modify the parameters as needed. The main parameters are discussed in this chapter.

How to do it...

The config file has sections specified by `section_name`. All the parameters related to a section can be put under them, for example:

```
[mysqld] <---section name
<parameter_name> = <value> <---parameter values
[client]
<parameter_name> = <value>
[mysqldump]
<parameter_name> = <value>
[mysqld_safe]
<parameter_name> = <value>
[server]
<parameter_name> = <value>
```

- `[mysql]`: Section is read by the `mysql` command-line client
- `[client]`: Section is read by all connecting clients (including `mysql cli`)
- `[mysqld]`: Section is read by the `mysql` server
- `[mysqldump]`: The section is read by the backup utility called `mysqldump`
- `[mysqld_safe]`: Read by the `mysqld_safe` process (MySQL Server Startup Script)

Apart from that the `mysqld_safe` process reads all options from the `[mysqld]` and `[server]` sections in option files.

For example, `mysqld_safe` process reads the `pid-file` option from `mysqld` section.

```
shell> sudo vi /etc/my.cnf
[mysqld]
pid-file = /var/lib/mysql/mysqld.pid
```

In systems that use `systemd`, `mysqld_safe` will not be installed. To configure the startup script, you need to set the values in `/etc/systemd/system/mysqld.service.d/override.conf`.

For example:

```
[Service]
LimitNOFILE=max_open_files
PIDFile=/path/to/pid/file
LimitCore=core_file_limit
Environment="LD_PRELOAD=/path/to/malloc/library"
Environment="TZ=time_zone_setting"
```

Using global and session variables

As you have seen in the previous chapters, you can set the parameters by connecting to MySQL and executing the `SET` command.

There are two types of variables based on the scope of the variable:

- **Global**: Applies to all the new connections
- **Session**: Applies only to the current connection (session)

How to do it...

For example, if you want to log all queries that are slower than one second, you can execute:

```
mysql> SET GLOBAL long_query_time = 1;
```

To make the changes persistent across restarts use:

```
mysql> SET PERSIST long_query_time = 1;
Query OK, 0 rows affected (0.01 sec)
```

Or:

```
mysql> SET @@persist.long_query_time = 1;
Query OK, 0 rows affected (0.00 sec)
```

The persisted global system variable settings are stored in mysqld-auto.cnf which is located in data directory.

Suppose you want to log queries only for this session and not for all the connections. You can use the following command:

```
mysql> SET SESSION long_query_time = 1;
```

Using parameters with startup script

Suppose you wish to start MySQL using a startup script and not through `systemd`, especially for testing or for some temporary change. You can pass the variables to the script rather than changing it in the config file.

How to do it...

```
shell> /usr/local/mysql/bin/mysqld --
basedir=/usr/local/mysql --datadir=/usr/local/mysql/data --
plugin-dir=/usr/local/mysql/lib/plugin --user=mysql --log-
error=/usr/local/mysql/data/centos7.err --pid-
file=/usr/local/mysql/data/centos7.pid --init-
file=/tmp/mysql-init &
```

You can see that the `--init-file` parameter is passed to the server. The server executes the SQL statements in that file before starting.

Configuring the parameters

After installation, the basic things you need to configure are covered in this section. The rest all can be left as default or tuned later according to the load.

How to do it...

Let's get into the details.

data directory

Data managed by the MySQL server is stored under a directory known as the `data directory`. Each sub-directory of the `data directory` is a database directory and corresponds to a database managed by the server. By default, the

`data directory` has three sub directories:

- `mysql`: MySQL system database
- `performance_schema`: Provides information used to inspect the internal execution of the server at runtime
- `sys`: Provides a set of objects to help interpret performance schema information more easily

Apart from these, the `data directory` contains the log files, `InnoDB` tablespace and `InnoDB` log files, SSL and RSA key files, `pid` of `mysqld`, and `mysqld-auto.cnf`, which stores persisted global system variable settings.

To set the `data directory` change/add the value of `datadir` to the config file. The default is `/var/lib/mysql`:

```
shell> sudo vi /etc/my.cnf
[mysqld]
datadir = /data/mysql
```

You can set it to wherever you want to store the data, but you should change the ownership of the `data directory` to `mysql`.

TIP *Make sure that the disk volume bearing the `data directory` has sufficient space to hold all your* *data.*

innodb_buffer_pool_size

This is the most important tuning parameter that decides how much memory the `InnoDB` storage engine can use to cache data and indexes in memory. Setting it too low can degrade the performance of the MySQL server, and setting it too high can increase the memory consumption of MySQL process. The best thing about MySQL 8 is that `innodb_buffer_pool_size` is dynamic, meaning you can vary `innodb_buffer_pool_size` without restarting the server.

Here is a simple guide on how to tune it:

1. Find out the size of your dataset. Do not set the value of `innodb_buffer_pool_size` higher than that of your dataset. Suppose you have a 12 GB RAM machine and your dataset is 3 GB; then you can set `innodb_buffer_pool_size` to 3 GB. If you expect growth in your data, you can increase it as and when needed without restarting MySQL.

2. Usually, the size of the dataset is much bigger than the available RAM. Out of the total RAM, you can set some for the operating system, some for other processes, some for per-thread buffers inside MySQL, and some for the MySQL server apart from `InnoDB`. The rest can be assigned to the `InnoDB` buffer pool size.
   This is a very generic table and gives you a good value to start with, assuming that it is a dedicated MySQL server, all the tables are `InnoDB`, and per-thread buffers are left as default. If the system is running out of memory, you can decrease the buffer pool dynamically.

RAM	**Buffer Pool Size (Range)**
4 GB	1 GB-2 GB
8 GB	4 GB-6 GB
12 GB	6 GB-10 GB
16 GB	10 GB-12 GB
32 GB	24 GB-28 GB
64 GB	45 GB-56 GB
128 GB	108 GB-116 GB
256 GB	220 GB-245 GB

innodb_buffer_pool_instances

You can divided the `InnoDB` buffer pool into separate regions to improve concurrency, by reducing contention as different threads read and write to cached pages. For example, if the buffer pool size is 64 GB and `innodb_buffer_pool_instances` are 32, the buffer is split into 32 regions with 2 GB each.

If the buffer pool size is more than 16 GB, you can set the instances so that each region gets at least 1 GB of space.

innodb_log_file_size

This is the size of the redo log space used to replay committed transactions in case of a database crash. The default is 48 MB, which may not be sufficient for production workloads. To start with, you can set 1 GB or 2 GB. This change requires a restart. Stop the MySQL server and make sure that it shuts down without errors. Make the changes in `my.cnf` and start the server. In earlier versions, you need to stop the server, remove the log files, and then start the server. In MySQL 8, it is automatic. Modifying the redo log files is explained in Chapter 11, *Managing Tablespace*, in the *Changing the number or size of InnoDB redo log files* section.

Changing the data directory

Your data can grow over time, and when it outgrows the filesystem, you need to add a disk or move the `data directory` to a bigger volume.

How to do it...

1. Check the current `data directory`. By default, the `data directory` is `/var/lib/mysql`:

```
mysql> show variables like '%datadir%';
+---------------+-----------------+
| Variable_name | Value           |
+---------------+-----------------+
| datadir       | /var/lib/mysql/ |
+---------------+-----------------+
1 row in set (0.04 sec)
```

2. Stop `mysql` and make sure it has stopped successfully:

```
shell> sudo systemctl stop mysql
```

3. Check the status:

```
shell> sudo systemctl status mysql
```

   It should show `Stopped MySQL Community Server`.

4. Create the directory at the new location and change the ownership to `mysql`:

```
shell> sudo mkdir -pv /data
shell> sudo chown -R mysql:mysql /data/
```

5. Move the files to the new `data directory`:

```
shell> sudo rsync -av /var/lib/mysql /data
```

6. In Ubuntu, if you've enabled AppArmor, you need to configure the Access Control:

```
shell> vi /etc/apparmor.d/tunables/alias
alias /var/lib/mysql/ -> /data/mysql/,
shell> sudo systemctl restart apparmor
```

7. Start MySQL server and verify that the `data` directory has changed:

```
shell> sudo systemctl start mysql
mysql> show variables like '%datadir%';
+---------------+--------------+
| Variable_name | Value        |
+---------------+--------------+
| datadir       | /data/mysql/ |
+---------------+--------------+
1 row in set (0.00 sec)
```

8. Verify that the data is intact and remove the old `data directory`:

```
shell> sudo rm -rf /var/lib/mysql
```

If MySQL fails to starts with the error—`MySQL data dir not found at /var/lib/mysql, please create one`:

Execute, `sudo mkdir /var/lib/mysql/mysql -p`

If it says `MySQL system database not found`, run the ;`mysql_install_db` tool, which creates the required directories.

Transactions

In this chapter, we will cover the following recipes:

- Performing transactions
- Using savepoints
- Isolation levels
- Locking

Introduction

In the following recipes, we will discuss the transactions and various isolation levels in MySQL. Transaction means a set of SQL statements that should succeed or fail together. Transactions should also satisfy **Atomicity, Consistency, Isolation, and Durability**(**ACID**) properties. Take a very basic example of a money transfer from account `A` to account `B`. Assume that `A` has $600, `B` has $400, and `B` wishes to transfer $100 from `A` to itself.

The bank would deduct $100 from `A` and add to `B` using the following SQL code (for illustration):

```
mysql> SELECT balance INTO @a.bal FROM account WHERE
account_number='A';
```

Programmatically, check whether `@a.bal` is greater than or equal to 100:

```
mysql> UPDATE account SET balance=@a.bal-100 WHERE
account_number='A';
mysql> SELECT balance INTO @b.bal FROM account WHERE
account_number='B';
```

Programmatically, check whether `@b.bal` is `NOT NULL`:

```
mysql> UPDATE account SET balance=@b.bal+100 WHERE
account_number='B';
```

These four SQL lines should be part of a single transaction and satisfy the following ACID properties:

- **Atomicity**: Either all the SQLs should be successful or all should fail. There should not be any partial updates. If this property is not obeyed and if the database crashes after running two SQLs, then `A` would lose 100.
- **Consistency**: A transaction must change affected data only in allowed ways. In this example, if `account_number` with `B` does not exist, the whole transaction should be rolled back.
- **Isolation**: Transactions that occur at the same time (concurrent transactions) should not lead the database to an inconsistent state. Each of the transactions should be executed as if it were the only transaction in the system. No transaction should affect the existence of any other transaction. Suppose `A` transfers all of this 600 exactly at the same time while transferring to `B`; both transactions should act independently, ensuring the balance before transferring the amount.
- **Durability**: The data should be persistent on disk and should not be lost despite any database or system failure.

`InnoDB`, the default storage engine in MySQL, supports transactions whereas MyISAM does not support them.

Performing transactions

Create dummy tables and sample data to understand this recipe:

```
mysql> CREATE DATABASE bank;
mysql> USE bank;
mysql> CREATE TABLE account(account_number varchar(10)
PRIMARY KEY, balance int);
mysql> INSERT INTO account VALUES('A',600),('B',400);
```

How to do it...

To start a transaction (set of SQLs), execute the `START TRANSACTION` or `BEGIN` statement:

```
mysql> START TRANSACTION;
or
mysql> BEGIN;
```

Then execute all the statements that you wish to be inside a transaction, such as transferring 100 from `A` to `B`:

```
mysql> SELECT balance INTO @a.bal FROM account WHERE
account_number='A';

Programmatically check if @a.bal is greater than or equal to
100
mysql> UPDATE account SET balance=@a.bal-100 WHERE
account_number='A';
mysql> SELECT balance INTO @b.bal FROM account WHERE
account_number='B';

Programmatically check if @b.bal IS NOT NULL
mysql> UPDATE account SET balance=@b.bal+100 WHERE
account_number='B';
```

After making sure that all the SQLs are executed successfully, execute the `COMMIT` statement, which will finish the transaction and commit the data:

```
mysql> COMMIT;
```

If you encounter any error in between and wish to abort the transaction, you can issue a `ROLLBACK` statement instead of `COMMIT`.

For example, instead of sending to `B`, if `A` wants to transfer to an account that does not exist, you should abort the transaction and refund the amount to `A`:

```
mysql> BEGIN;

mysql> SELECT balance INTO @a.bal FROM account WHERE
account_number='A';

mysql> UPDATE account SET balance=@a.bal-100 WHERE
account_number='A';

mysql> SELECT balance INTO @b.bal FROM account WHERE
account_number='C';
Query OK, 0 rows affected, 1 warning (0.07 sec)

mysql> SHOW WARNINGS;
+---------+------+-----------------------------------------------
-----------+
| Level   | Code | Message
|
+---------+------+-----------------------------------------------
-----------+
| Warning | 1329 | No data - zero rows fetched, selected, or
processed |
+---------+------+-----------------------------------------------
-----------+
1 row in set (0.02 sec)

mysql> SELECT @b.bal;
+--------+
| @b.bal |
+--------+
| NULL   |
+--------+
1 row in set (0.00 sec)

mysql> ROLLBACK;
Query OK, 0 rows affected (0.01 sec)
```

Autocommit

By default, autocommit is `ON`, which means that all individual statements are committed as soon as they are executed unless they are in a `BEGIN...COMMIT` block. If autocommit is `OFF`, you need to explicitly issue a `COMMIT` statement to commit a transaction. To disable it, execute:

```
mysql> SET autocommit=0;
```

DDL statements, such as `CREATE` or `DROP` for databases and `CREATE`, `DROP`, or `ALTER` for tables or stored routines cannot be rolled back.

TIP *There are certain statements such as DDLs, `LOAD DATA INFILE`, `ANALYZE TABLE`, replication-related statements and so on that cause implicit `COMMIT`. For more details on these statements, refer https://dev.mysql.com/doc/refman/8.0/en/implicit-commit.html.*

Using savepoints

Using savepoints, you can roll back to certain points in the transaction without terminating the transaction. You can use `SAVEPOINT identifier` to set a name for the transaction and use the `ROLLBACK TO identifier` statement to roll back a transaction to the named savepoint without terminating the transaction.

How to do it...

Suppose `A` wants to transfer to multiple accounts; even if a transfer to one account fails, the others should not be rolled back:

```
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT balance INTO @a.bal FROM account WHERE
account_number='A';
Query OK, 1 row affected (0.01 sec)

mysql> UPDATE account SET balance=@a.bal-100 WHERE
account_number='A';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> UPDATE account SET balance=balance+100 WHERE
account_number='B';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> SAVEPOINT transfer_to_b;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT balance INTO @a.bal FROM account WHERE
account_number='A';
Query OK, 1 row affected (0.00 sec)

mysql> UPDATE account SET balance=balance+100 WHERE
account_number='C';
Query OK, 0 rows affected (0.00 sec)
Rows matched: 0 Changed: 0 Warnings: 0

### Since there are no rows updated, meaning there is no
account with 'C', you can rollback the transaction to
SAVEPOINT where transfer to B is successful. Then 'A' will
get back 100 which was deducted to transfer to C. If you
wish not to use the save point, you should do these in two
transactions.
```

```
mysql> ROLLBACK TO transfer_to_b;
Query OK, 0 rows affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT balance FROM account WHERE account_number='A';
+---------+
| balance |
+---------+
| 400     |
+---------+
1 row in set (0.00 sec)

mysql> SELECT balance FROM account WHERE account_number='B';
+---------+
| balance |
+---------+
| 600     |
+---------+
1 row in set (0.00 sec)
```

Isolation levels

When two or more transactions occur at the same time, the isolation level defines the degree at which a transaction is isolated from the resource or data modifications made by other transactions. There are four types of isolation levels; to change the isolation level, you need to set the `tx_isolation` variable which is dynamic and has session level scope.

How to do it...

To change this level, execute `SET @@transaction_isolation = 'READ-COMMITTED';`.

Read uncommitted

The current transaction can read data written by another uncommitted transaction, which is also called **dirty read**.

For example, `A` wants to add some amount to his account and transfer it to `B`. Assume both the transactions happen at the same time; the flow will be like this.

`A` initially has $400 and wants to transfer $500 to `B` after adding $500 to his account.

# Transaction 1 (adding amount)	# Transaction 2 (transferring amount)
`BEGIN;`	`BEGIN;`

`UPDATE account`
` SET balance=balance+500`
` WHERE account_number='A';`
	--

--	`SELECT balance INTO @a.bal`
	` FROM account`
	` WHERE account_number='A';`
	` # A sees 900 here`

`ROLLBACK;`
` # Assume due to some reason the`
` transaction got rolled back`
	--

--	`# A transfers 900 to B since`
	` A has 900 in previous SELECT`
	` UPDATE account`
	` SET balance=balance-900`
	` WHERE account_number='A';`

--	`# B receives the amount`
	`UPDATE account`
	` SET balance=balance+900`
	` WHERE account_number='B';`

--	`# Transaction 2 completes successfully`
	`COMMIT;`

You can notice that *Transaction 2* has read the data that is not committed or rolled back from *Transaction 1*, causing account `A` to go into negative balance after this transaction, which is clearly not desired.

Read committed

The current transaction can read only the data committed by another transaction, which is also called **non-repeatable read**.

Take the same example again where `A` has $400 and `B` has $600.

# Transaction 1 (adding amount)	**# Transaction 2 (transferring amount)**
`BEGIN;`	`BEGIN;`

`UPDATE account SET`
`balance=balance+500`
`WHERE account_number='A';`
	--

--	`SELECT balance INTO @a.bal`
	`FROM account`
	`WHERE account_number='A';`
	`# A sees 400 here because transaction 1 has not committed the data yet`

`COMMIT;`	--

--	`SELECT balance INTO @a.bal`
	`FROM account`
	`WHERE account_number='A';`
	`# A sees 900 here because transaction 1 has committed the data.`

You can notice that, in the same transaction, different results are fetched for the same `SELECT` statement.

Repeatable read

A transaction will see the same data that is read by the first statement even though another transaction has committed the data. All consistent reads within the same transaction read the snapshot established by the first read. An exception is a transaction that can read the data changed within the same transaction.

When a transaction starts and executes its first read, a read view will be created and stay open until the end of the transaction. In order to provide the same result set until the end of the transaction, `InnoDB` uses row versioning and `UNDO` information. Suppose *Transaction 1* has selected a few rows and another transaction has deleted those rows and committed the data. If *Transaction 1* is open, it should be able to see the rows it has selected at the beginning. The deleted rows are preserved in the `UNDO` log space to fulfill *Transaction 1*. Once *Transaction 1* finishes, the rows are marked to be deleted from the `UNDO` logs. This is called **Multi-Version Concurrency Control** (**MVCC**).

Take the same example again where `A` has 400 and `B` has 600.

# Transaction 1 (adding the amount)	# Transaction 2 (transferring the amount)
`BEGIN;`	`BEGIN;`

--	`SELECT balance INTO @a.bal`
	`FROM account`
	`WHERE account_number='A';`
	`# A sees 400 here`

`UPDATE account`
`SET balance=balance+500`
	--

`WHERE account_number='A';`	

--	`SELECT balance INTO @a.bal`
	`FROM account`
	`WHERE account_number='A';`
	`# A sees still 400 even though transaction 1 is committed`

`COMMIT;`	--
--	`COMMIT;`

--	`SELECT balance INTO @a.bal`
	`FROM account`
	`WHERE account_number='A';`
	`# A sees 900 here because this is a fresh transaction`

This applies only to `SELECT` statements and not necessarily to DML statements. If you insert or modify some rows and then commit that transaction, a `DELETE` or `UPDATE` statement issued from another concurrent `REPEATABLE READ` transaction could affect those just-committed rows, even though the session cannot query them. If a transaction does update or delete rows committed by a different transaction, those changes become visible to the current transaction.

For example:

# Transaction 1	# Transaction 2
`BEGIN;`	`BEGIN;`

`SELECT * FROM account;`
`# 2 rows are returned`
	--

--	`INSERT INTO account VALUES('C',1000);`
	`# New account is created`

--	COMMIT;
SELECT * FROM account WHERE account_number='C'; **# no rows are returned because of MVCC**	--
DELETE FROM account WHERE account_number='C'; **# Surprisingly account C gets deleted**	--
--	SELECT * FROM account; **# 3 rows are returned because transaction 1 is not yet committed**
COMMIT;	--
--	SELECT * FROM account; **# 2 rows are returned because transaction 1 is committed**

Here's another example:

# Transaction 1	# Transaction 2
BEGIN;	BEGIN;
SELECT * FROM account; **# 2 rows are returned**	--
--	INSERT INTO account VALUES('D',1000);
--	COMMIT;
SELECT * FROM account; **# 3 rows are returned because of MVCC**	--
UPDATE account SET balance=1000 WHERE	--

`account_number='D';`
`#` **`Surprisingly account D gets updated`**	

`SELECT * FROM account;`
`# Surprisingly 4 rows are returned`
	--

Serializable

This provides the highest level of isolation by locking all the rows that are being selected. This level is like `REPEATABLE READ`, but `InnoDB` implicitly converts all plain `SELECT` statements to `SELECT...LOCK IN SHARE MODE` if autocommit is disabled. If autocommit is enabled, `SELECT` is its own transaction.

For example:

# Transaction 1	**# Transaction 2**
`BEGIN;`	`BEGIN;`
`SELECT * FROM account WHERE account_number='A';`	--
--	`UPDATE account SET balance=1000 WHERE account_number='A';` **`# This will wait until the lock held by transaction 1 on row A is released`**
`COMMIT;`	--
--	**`# UPDATE will be successful now`**

Another example:

# Transaction 1	**# Transaction 2**
`BEGIN;`	`BEGIN;`
`SELECT * FROM account WHERE account_number='A';`	--

# Selects values of A	
--	INSERT INTO account VALUES('D',2000); **# Inserts D**
SELECT * FROM account WHERE account_number='D'; # This will wait until the transaction 2 completes	--
--	COMMIT;
# Now the preceding select statement returns values of D	--

So, serializable waits for the locks and always reads the latest committed data.

Locking

There are two types of locking:

- **Internal locking**: MySQL performs internal locking within the server itself to manage contention for table contents by multiple sessions
- **External locking**: MySQL gives the option to client sessions to acquire a table lock explicitly for preventing other sessions from accessing the table

Internal locking: There are mainly two types of locks:

- **Row-level locks**: The locking is granular to the level of rows. Only the rows that are accessed are locked. This allows simultaneous write access by multiple sessions, making them suitable for multi-user, highly concurrent, and OLTP applications. Only `InnoDB` supports row-level locks.
- **Table-level locks**: MySQL uses table-level locking for `MyISAM`, `MEMORY`, and `MERGE` tables, permitting only one session to update those tables at a time. This locking level makes these storage engines more suitable for read-only, read-mostly, or single-user applications.

Refer to https://dev.mysql.com/doc/refman/8.0/en/internal-locking.html and https://dev.mysql.com/doc/refman/8.0/en/innodb-locking.html to know more about the `InnoDB` locks.

External Locking: You can use `LOCK TABLE` and `UNLOCK TABLES` statements to control the locks.

Table locking for `READ` and `WRITE` are explained as follows:

- `READ`: When a table is locked for `READ`, multiple sessions can read data from the table without acquiring a lock. Also, multiple sessions can acquire a lock on the same table, that is why a `READ` lock is also called a **shared lock**. When a `READ` lock is held, no session can write data into the table (including the session that held the lock). If any write attempt is make, it will be in waiting state until the `READ` lock is released.
- `WRITE`: When a table is locked for `WRITE`, no other session can read and write data from the table except the session that held the lock. No other session can even acquire any lock until the existing lock is released. That is why this is called `exclusive lock`. If any read/write attempt is make, it will be in waiting state until the `WRITE` lock is released.

All the locks are released when the ;`UNLOCK TABLES` statement is executed or when the session terminates.

How to do it...

The syntax is as follows:

```
mysql> LOCK TABLES table_name [READ | WRITE]
```

To unlock the tables, use:

```
mysql> UNLOCK TABLES;
```

To lock all tables across all databases, execute the following statement. It is used when taking a consistent snapshot of the database. It freezes all writes to the database:

```
mysql> FLUSH TABLES WITH READ LOCK;
```

Locking queue

No two locks can be held together on a table except shared locks (a table can have multiple shared locks). If a table already has a shared lock and an exclusive lock comes, it will be kept in a queue until the shared lock is released. When an exclusive lock is in a queue, all subsequent shared locks are also blocked and kept in a queue.

`InnoDB` acquires a metadata lock when reading/writing from a table. If a second transaction requests `WRITE LOCK`, it will be kept in a queue until the first transaction completes. If a third transaction wants to read the data, it has to wait until the second transaction completes.

Transaction 1:

```
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM employees LIMIT 10;
+--------+------------+------------+-----------+--------+---
---------+
| emp_no | birth_date | first_name | last_name | gender |
hire_date  |
+--------+------------+------------+-----------+--------+---
---------+
|  10001 | 1953-09-02 | Georgi     | Facello   | M      |
1986-06-26 |
|  10002 | 1964-06-02 | Bezalel    | Simmel    | F      |
1985-11-21 |
|  10003 | 1959-12-03 | Parto      | Bamford   | M      |
1986-08-28 |
|  10004 | 1954-05-01 | Chirstian  | Koblick   | M      |
1986-12-01 |
|  10005 | 1955-01-21 | Kyoichi    | Maliniak  | M      |
1989-09-12 |
|  10006 | 1953-04-20 | Anneke     | Preusig   | F      |
1989-06-02 |
|  10007 | 1957-05-23 | Tzvetan    | Zielinski | F      |
1989-02-10 |
|  10008 | 1958-02-19 | Saniya     | Kalloufi  | M      |
1994-09-15 |
|  10009 | 1952-04-19 | Sumant     | Peac      | F      |
1985-02-18 |
|  10010 | 1963-06-01 | Duangkaew  | Piveteau  | F      |
1989-08-24 |
+--------+------------+------------+-----------+--------+---
---------+
10 rows in set (0.00 sec)
```

Note that `COMMIT` is not executed. The transaction is kept open.

Transaction 2:

```
mysql> LOCK TABLE employees WRITE;
```

This statement has to wait until transaction 1 completes.

Transaction 3:

```
mysql> SELECT * FROM employees LIMIT 10;
```

Even transaction 3 will not give any result because an exclusive lock is in the queue (it is waiting for transaction 2 to complete). Moreover, it is blocking all operations on the table.

You can check this by checking `SHOW PROCESSLIST` from another session:

```
mysql> SHOW PROCESSLIST;
+----+------+-----------+-----------+---------+------+------
----------------------------+-------------------------------
--+
| Id | User | Host      | db        | Command | Time | State
| Info                          |
+----+------+-----------+-----------+---------+------+------
---------------------------+-------------------------------
--+
| 20 | root | localhost | employees | Sleep   |   48 |
| NULL                            |
| 21 | root | localhost | employees | Query   |   34 |
Waiting for table metadata lock | LOCK TABLE employees WRITE
|
| 22 | root | localhost | employees | Query   |   14 |
Waiting for table metadata lock | SELECT * FROM employees
LIMIT 10 |
| 23 | root | localhost | employees | Query   |    0 |
starting                        | SHOW PROCESSLIST
|
+----+------+-----------+-----------+---------+------+------
---------------------------+-------------------------------
--+
4 rows in set (0.00 sec)
```

You can notice that both transaction 2 and transaction 3 are waiting for transaction 1.

To know more about a metadata lock, refer to https://dev.mysql.com/doc/refman/8.0/en/metadata-locking.html. The same behavior can be observed while using `FLUSH TABLES WITH READ LOCK`.

Transaction 1:

```
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM employees LIMIT 10;
+--------+------------+------------+-----------+--------+---
---------+
| emp_no | birth_date | first_name | last_name | gender |
hire_date  |
+--------+------------+------------+-----------+--------+---
---------+
|  10001 | 1953-09-02 | Georgi     | Facello   | M      |
1986-06-26 |
|  10002 | 1964-06-02 | Bezalel    | Simmel    | F      |
1985-11-21 |
|  10003 | 1959-12-03 | Parto      | Bamford   | M      |
1986-08-28 |
|  10004 | 1954-05-01 | Chirstian  | Koblick   | M      |
1986-12-01 |
|  10005 | 1955-01-21 | Kyoichi    | Maliniak  | M      |
1989-09-12 |
|  10006 | 1953-04-20 | Anneke     | Preusig   | F      |
1989-06-02 |
|  10007 | 1957-05-23 | Tzvetan    | Zielinski | F      |
1989-02-10 |
|  10008 | 1958-02-19 | Saniya     | Kalloufi  | M      |
1994-09-15 |
|  10009 | 1952-04-19 | Sumant     | Peac      | F      |
1985-02-18 |
|  10010 | 1963-06-01 | Duangkaew  | Piveteau  | F      |
1989-08-24 |
+--------+------------+------------+-----------+--------+---
---------+
10 rows in set (0.00 sec)
```

Note that `COMMIT` is not executed. The transaction is kept open.

Transaction 2:

```
mysql> FLUSH TABLES WITH READ LOCK;
```

Transaction 3:

```
mysql> SELECT * FROM employees LIMIT 10;
```

Even transaction 3 will not give any result because `FLUSH TABLES` is waiting for all the operations on the table to complete before getting a lock. Moreover, it is blocking all the operations on the table.

You can check this by checking `SHOW PROCESSLIST` from another session.

```
mysql> SHOW PROCESSLIST;
+----+------+-----------+----------+---------+------+------
```

```
------------------+----------------------------------------
----------+
| Id | User | Host      | db        | Command | Time | State
| Info                                             |
+----+------+-----------+-----------+---------+------+------
------------------+----------------------------------------
----------+
| 20 | root | localhost | employees | Query   |    7 |
Creating sort index     | SELECT * FROM employees ORDER BY
first_name DESC |
| 21 | root | localhost | employees | Query   |    5 |
Waiting for table flush | FLUSH TABLES WITH READ LOCK
|
| 22 | root | localhost | employees | Query   |    3 |
Waiting for table flush | SELECT * FROM employees LIMIT 10
|
| 23 | root | localhost | employees | Query   |    0 |
starting                | SHOW PROCESSLIST
|
+----+------+-----------+-----------+---------+------+------
------------------+----------------------------------------
----------+
4 rows in set (0.00 sec)
```

For consistent backup, all the backup methods use `FLUSH TABLES WITH READ LOCK`, which can be very dangerous if there is a long-running transaction on table.

Binary Logging

In this chapter, we will cover the following recipes:

- Using binary logging
- Binary log format
- Extracting statements from a binary log
- Ignoring databases to write to a binary log
- Relocating binary logs

Introduction

The binary log contains a record of all changes to the database, both data and structure. The binary log is not used for statements such as `SELECT` or `SHOW` that do not modify data. Running a server with binary logging enabled has a slight performance effect. The binary log is crash-safe. Only complete events or transactions are logged or read back.

Why should you use binary log?

- **Replication**: You stream the changes made to a server to another server using binary logs. The slave acts as a mirror copy and can be used to distribute the load. The server that accepts the writes is referred to as a master and the mirror copy server is referred to as a slave.
- **Point-in-time recovery**: Suppose you take a backup at 00:00 on Sunday and your database crashed at 8:00 on Sunday. Using backup, you can recover till 00:00 Sunday. Using binary logs you can reply to them, to recover till 08:00.

Using binary logging

To enable `binlog`, you have to set `log_bin` and `server_id` and restart the server. You can mention the path and base name in the `log_bin` itself. For example, `log_bin` is set to `/data/mysql/binlogs/server1`, the binary logs are stored in the `/data/mysql/binlogs` folder with the name `server1.000001`, `server1.000002`, and so on. The server creates a new file in the series each time it starts or flushes the logs or the current log's size reaches `max_binlog_size`. It maintains the `server1.index` file, which contains the location of each binary log.

How to do it...

Let's see how to play with the logs. I am sure you are going to love learning about them.

Enabling binary logs

1. Enable binary logging and set the `server_id`. Open the MySQL `config` file in your favorite editor and append the following lines. Choose `server_id` such that it will be unique to each MySQL server in your infrastructure.
   You can also simply put the `log_bin` variable in `my.cnf` without any value. In that case, the binary log is created in the `data directory` directory and uses `hostname` as its name.

```
shell> sudo vi /etc/my.cnf
[mysqld]
log_bin = /data/mysql/binlogs/server1
server_id = 100
```

2. Restart the MySQL server:

```
shell> sudo systemctl restart mysql
```

3. Verify that binary logs are created:

```
mysql> SHOW VARIABLES LIKE 'log_bin%';
+----------------------------------+--------------------
---------------+
| Variable_name                    | Value
|
+----------------------------------+--------------------
---------------+
| log_bin                          | ON
|
| log_bin_basename                 |
/data/mysql/binlogs/server1       |
| log_bin_index                    |
/data/mysql/binlogs/server1.index |
| log_bin_trust_function_creators | OFF
|
| log_bin_use_v1_row_events       | OFF
```

```
|
+----------------------------------+-------------------
---------------+
5 rows in set (0.00 sec)
```

```
mysql> SHOW MASTER LOGS;
+----------------+-----------+
| Log_name       | File_size |
+----------------+-----------+
| server1.000001 | 154       |
+----------------+-----------+
1 row in set (0.00 sec)
```

```
shell> sudo ls -lhtr /data/mysql/binlogs
total 8.0K
-rw-r----- 1 mysql mysql 34 Aug 15 05:01 server1.index
-rw-r----- 1 mysql mysql 154 Aug 15 05:01
server1.000001
```

4. Execute `SHOW BINARY LOGS;` or `SHOW MASTER LOGS;` to display all the binary logs of the server.
5. Execute the `SHOW MASTER STATUS;` command to get the current binary log position:

```
mysql> SHOW MASTER STATUS;
+----------------+----------+--------------+----------
-------+-----------------
+++++++++++++++++++++++++++++++++++++-+
| File           | Position | Binlog_Do_DB |
Binlog_Ignore_DB | Executed_Gtid_Set |
+----------------+----------+--------------+----------
-------+-------------------+
| server1.000002 |     3273 |              |
|                   |
+----------------+----------+--------------+----------
-------+-------------------+
1 row in set (0.00 sec)
```

As soon as `server1.000001` reaches `max_binlog_size` (1 GB default), a new file, `server1.000002`, will be created and added to `server1.index`. You can configure to set `max_binlog_size` dynamically with `SET @@global.max_binlog_size=536870912`.

Disabling binary logs for a session

There may be cases where you do not want the statements to be replicated to other servers. For that, you can use the following command to disable binary logging for that session:

```
mysql> SET SQL_LOG_BIN = 0;
```

All the SQL statements logged after executing the previous statement are not logged to the binary log. This applies only for that session.

To enable, you can execute the following:

```
mysql> SET SQL_LOG_BIN = 1;
```

Move to the next log

You can use the `FLUSH LOGS` command to close the current binary log and open a new one:

```
mysql> SHOW BINARY LOGS;
+----------------+-----------+
| Log_name       | File_size |
+----------------+-----------+
| server1.000001 |       154 |
+----------------+-----------+
1 row in set (0.00 sec)

mysql> FLUSH LOGS;
Query OK, 0 rows affected (0.02 sec)

mysql> SHOW BINARY LOGS;
+----------------+-----------+
| Log_name       | File_size |
+----------------+-----------+
| server1.000001 |  198      |
| server1.000002 |  154      |
+----------------+-----------+
2 rows in set (0.00 sec)
```

Expire binary logs

Binary logs consume a lot of space based on the number of writes. Leaving them as-is can fill up the disk in no time. It is essential to clean them up:

1. Set the expiry of logs using `binlog_expire_logs_seconds` and `expire_logs_days`.
   If you want to set an expiry period in days, set `expire_logs_days`. For example, if you want to delete all the binary logs that are older than two days, `SET @@global.expire_logs_days=2`. Setting the value to `0` disables automatic expiry.
   If you want to have more granularity, you can use the `binlog_expire_logs_seconds` variable, which sets the binary log expiration period in seconds.
   The effects of this variable and `expire_logs_days` are cumulative. For example, if `expire_logs_days` is `1` and `binlog_expire_logs_seconds` is `43200`, then the binary log is purged every 1.5 days. This produces the same result as setting `binlog_expire_logs_seconds` to `129600` and leaving `expire_logs_days` set to 0. In MySQL 8.0, both `binlog_expire_logs_seconds` and `expire_logs_days` must be set to 0 to disable automatic purging of the binary log.
2. To purge the logs manually, execute `PURGE BINARY LOGS TO '<file_name>'`. For example, there are files such as `server1.000001`, `server1.000002`, `server1.000003`, and `server1.000004`, and if you execute `PURGE BINARY LOGS TO 'server1.000004'`, all the files up to `server1.000003` will be deleted and `server1.000004` will not be touched:

```
mysql> SHOW BINARY LOGS;
+----------------+-----------+
| Log_name       | File_size |
+----------------+-----------+
| server1.000001 |       198 |
| server1.000002 |       198 |
| server1.000003 |       198 |
| server1.000004 |       154 |
+----------------+-----------+
4 rows in set (0.00 sec)
```

```
mysql> PURGE BINARY LOGS TO 'server1.000004';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SHOW BINARY LOGS;
+----------------+-----------+
| Log_name       | File_size |
+----------------+-----------+
| server1.000004 |       154 |
+----------------+-----------+
1 row in set (0.00 sec)
```

You can also execute the command `PURGE BINARY LOGS BEFORE '2017-08-03 15:45:00'` rather than specifying a log file. You can also use the word `MASTER` instead of `BINARY`. `mysql> PURGE MASTER LOGS TO 'server1.000004'` also works.

3. To delete all the binary logs and start from the beginning again, execute `RESET MASTER`:

```
mysql> SHOW BINARY LOGS;
+----------------+-----------+
| Log_name       | File_size |
+----------------+-----------|
| server1.000004 |       154 |
+----------------+-----------+
1 row in set (0.00 sec)
```

```
mysql> RESET MASTER;
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> SHOW BINARY LOGS;
+----------------+-----------+
| Log_name       | File_size |
+----------------+-----------+
| server1.000001 |       154 |
+----------------+-----------+
1 row in set (0.00 sec)
```

Purging or expiring logs is a very unsafe method if you are using replication. The safe way to purge binary logs is to use the `mysqlbinlogpurge` script, which will be covered in Chapter 12*, Managing Logs.*

Binary log format

Binary logs can be written in three formats:

1. `STATEMENT`: Actual SQL statements are logged.
2. `ROW`: Changes made to each row are logged.
   For example, an update statement updates 10 rows, the updated information of all 10 rows is written to the log. Whereas in statement-based replication, only the update statement is written. The default format is `ROW`.
3. `MIXED`: MySQL switches from `STATEMENT` to `ROW` as and when needed.

There are statements that can cause different results when executed on different servers. For example, the output of the `UUID()` function differs from server to server. Those statements are called non-deterministic and are unsafe for statement-based replication. In those situations, a MySQL server switches to row-based format when you set the `MIXED` format.

Refer to https://dev.mysql.com/doc/refman/8.0/en/binary-log-mixed.html to learn more about the unsafe statements and when switching happens.

How to do it...

You can set the format using the dynamic variable, `binlog_format`, which has both global and session scope. Setting it at global level makes all clients use the specified format:

```
mysql> SET GLOBAL binlog_format = 'STATEMENT';
```

Or:

```
mysql> SET GLOBAL binlog_format = 'ROW';
```

Refer to https://dev.mysql.com/doc/refman/8.0/en/replication-sbr-rbr.html to learn about the advantages and disadvantages of various formats.

1. MySQL 8.0 uses version 2 binary log row events, which cannot be read by MySQL Server releases prior to MySQL 5.6.6. Set `log-bin-use-v1-row-events` to `1` to use version 1 so that it can be read by versions prior to MySQL 5.6.6. The default is `0`.

```
        mysql> SET @@GLOBAL.log_bin_use_v1_row_events=0;
```

2. When you create a stored function, you must declare either that it is deterministic or that it does not modify data. Otherwise, it may be unsafe for binary logging. By default, for a `CREATE FUNCTION` statement to be accepted, at least one of `DETERMINISTIC`, `NO SQL`, or `READS SQL DATA` must be specified explicitly. Otherwise an error occurs:

```
        ERROR 1418 (HY000): This function has none of
        DETERMINISTIC, NO SQL, or READS SQL DATA in its
```

```
      declaration and binary logging is enabled (you *might*
      want to use the less safe
      log_bin_trust_function_creators variable)
```

You can write non-deterministic statements inside a routine and still declare as `DETERMINISTIC` (not a good practice), if you want to replicate routines that are not declared as `DETERMINISTIC`, you can set the `log_bin_trust_function_creators` variable:

```
mysql> SET GLOBAL log_bin_trust_function_creators = 1;
```

See also

Refer to https://dev.mysql.com/doc/refman/8.0/en/stored-programs-logging.html to learn more about how the stored programs are replicated.

Extracting statements from a binary log

You can use the `mysqlbinlog` utility (shipped along with MySQL) to extract the contents from binary logs and apply them to other servers.

Getting ready

Execute a few statements using various binary formats. When you set the `binlog_format` at `GLOBAL` level, you have to disconnect and reconnect to get the changes. If you want to be connected, set at `SESSION` level.

Change to **statement-based replication** (**SBR**):

```
mysql> SET @@GLOBAL.BINLOG_FORMAT='STATEMENT';
Query OK, 0 rows affected (0.00 sec)
```

Update a few rows:

```
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE salaries SET salary=salary*2 WHERE
emp_no<10002;
Query OK, 18 rows affected (0.00 sec)
Rows matched: 18  Changed: 18  Warnings: 0

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

Change to **row-based replication** (**RBR**):

```
mysql> SET @@GLOBAL.BINLOG_FORMAT='ROW';
Query OK, 0 rows affected (0.00 sec)
```

Update a few rows:

```
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE salaries SET salary=salary/2 WHERE
emp_no<10002;
```

```
Query OK, 18 rows affected (0.00 sec)
Rows matched: 18  Changed: 18  Warnings: 0

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

Change to `MIXED` format:

```
mysql> SET @@GLOBAL.BINLOG_FORMAT='MIXED';
Query OK, 0 rows affected (0.00 sec)
```

Update a few rows:

```
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE salaries SET salary=salary*2 WHERE
emp_no<10002;
Query OK, 18 rows affected (0.00 sec)
Rows matched: 18  Changed: 18  Warnings: 0

mysql> INSERT INTO departments VALUES('d010',UUID());
Query OK, 1 row affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

How to do it...

To display the contents of `server1.000001`, execute the following:

```
shell> sudo mysqlbinlog /data/mysql/binlogs/server1.000001
```

You will get output similar to the following:

```
# at 226
#170815 12:49:24 server id 200  end_log_pos 312 CRC32
0x9197bf88  Query thread_id=5 exec_time=0 error_code=0
BINLOG '
~
~
```

In the first line, the number following `# at` indicates the starting position (file offset) of the event in the binary log file. The second line contains the timestamp at which the statement started on the server. The timestamp is followed by `server id`, `end_log_pos`, `thread_id`, `exec_time`, and `error_code`.

- `server id`: Is the `server_id` value of the server where the event originated (`200` in this case).
- `end_log_pos`: Is the start position of the next event.
- `thread_id`: Indicates which thread executed the event.
- `exec_time`: Is the time spent executing the event, on a master server. On a slave, it is the difference of the end execution time on the slave minus the beginning execution time on the master. The difference serves as an indicator of how much replication lags behind the master.
- `error_code`: Indicates the result from executing the event. Zero means that no error occurred.

Observations

1. You executed the UPDATE statement in statement-based replication and the same statement is logged in the binary log. Apart from the server, session variables are also saved in the binary log to replicate the same behavior on the slave:

```
# at 226
#170815 13:28:38 server id 200  end_log_pos 324 CRC32
0x9d27fc78  Query thread_id=8 exec_time=0 error_code=0
SET TIMESTAMP=1502803718/*!*/;
SET @@session.pseudo_thread_id=8/*!*/;
SET @@session.foreign_key_checks=1,
@@session.sql_auto_is_null=0,
@@session.unique_checks=1, @@session.autocommit=1/*!*/;
SET @@session.sql_mode=1436549152/*!*/;
SET @@session.auto_increment_increment=1,
@@session.auto_increment_offset=1/*!*/;
/*!\C utf8 *//*!*/;
SET
@@session.character_set_client=33,@@session.collation_c
onnection=33,@@session.collation_server=255/*!*/;
SET @@session.lc_time_names=0/*!*/;
SET @@session.collation_database=DEFAULT/*!*/;
```

```
BEGIN
/*!*/;
# at 324
#170815 13:28:38 server id 200  end_log_pos 471 CRC32
0x35c2ba45  Query thread_id=8 exec_time=0 error_code=0
use `employees`/*!*/;
SET TIMESTAMP=1502803718/*!*/;
UPDATE salaries SET salary=salary*2 WHERE emp_no<10002
/*!*/;
# at 471
#170815 13:28:40 server id 200  end_log_pos 502 CRC32
0xb84cfeda  Xid = 53
COMMIT/*!*/;
```

2. When row-based replication is used, instead of the statement, the `ROW` is saved, which is in binary format and you cannot read. Moreover, you can observe the length, a single update statement generated so much data. Check the *Extract row event display* section, which explains how to view the binary format.

```
BEGIN
/*!*/;
# at 660
#170815 13:29:02 server id 200  end_log_pos 722 CRC32
0xe0a2ec74  Table_map:`employees`.`salaries` mapped to
number 165
# at 722
#170815 13:29:02 server id 200  end_log_pos 1298 CRC32
0xf0ef8b05  Update_rows: table id 165 flags: STMT_END_F

BINLOG '
HveSWRPIAAAAPgAAANICAAAAAKUAAAAAAAAEACWVtcGxveWVlcwAIc2F
sYXJpZXMABAMDCgoAAAEBAHTsouA=HveSWR/IAAAAQAIAABIFAAAAAK
UAAAAAAAAEAAgAE///wEScAAFSrAwDahA/ahg/wEScAAKrVAQDahA/ah
g/wEScAAFjKAwDahg/ZiA/wEScAACzlAQDahg/ZiA/wEScAAGgIBADZ
iA/Zig/wEScAADQEAgDZiA/Zig/wEScAAJAQBADZig/ZjA/wEScAAEg
IAgDZig/ZjA/wEScAAEQWBADZjA/Zjg/wEScAACILAgDZjA/Zjg/wES
cAABhWBADZjg/YkA/wEScAAAwrAgDZjg/YkA/wEScAAHSJBADYkA/Yk
g/wEScAALpEAgDYkA/Ykg/wEScAAFiYBADYkg/YlA/wEScAACxMAgDY
kg/YlA/wEScAAGijBADYlA/Ylg/wEScAALRRAgDYlA/Ylg/wEScAAFC
xBADYlg/XmA/wEScAAKhYAgDYlg/XmA/wEScAADTiBADXmA/Xmg/wES
cAABpxAgDXmA/Xmg/wEScAAATyBADXmg/XnA/wEScAAAJ5AgDXmg/Xn
A/wEScAACTzBADXnA/Xng/wEScAAJJ5AgDXnA/Xng/wEScAANQuBQDX
ng/WoA/wEScAAGqXAgDXng/WoA/wEScAAOAxBQDWoA/Wog/wEScAAPC
YAgDWoA/Wog/wEScAAKQxBQDWog/WpA/wEScAANKYAgDWog/WpA/wES
cAAIAaBgDWpA8hHk7wEScAAEANAwDWpA8hHk7wEScAAIAaBgDSwg8hH
k7wEScAAEANAwDSwg8hHk4Fi+/w
'/*!*/;
# at 1298
#170815 13:29:02 server id 200  end_log_pos 1329 CRC32
0xa6dac5dc  Xid = 56
COMMIT/*!*/;
```

3. When `MIXED` format is used, the `UPDATE` statement is logged as SQL, whereas the `INSERT` statement is logged in row-based

format since the `INSERT` has an `UUID()` function that is non-deterministic:

```
BEGIN
/*!*/;
# at 1499
#170815 13:29:27 server id 200  end_log_pos 1646 CRC32
0xc73d68fb  Query thread_id=8 exec_time=0 error_code=0
SET TIMESTAMP=1502803767/*!*/;
UPDATE salaries SET salary=salary*2 WHERE emp_no<10002
/*!*/;
# at 1646
#170815 13:29:50 server id 200  end_log_pos 1715 CRC32
0x03ae0f7e  Table_map: `employees`.`departments` mapped
to number 166
# at 1715
#170815 13:29:50 server id 200  end_log_pos 1793 CRC32
0xa43c5dac  Write_rows: table id 166 flags: STMT_END_F
BINLOG
'TveSWRPIAAAARQAAALMGAAAAAKYAAAAAAAMACWVtcGxveWVlcwALZG
VwYXJ0bWVudHMAAv4PBP4QoAAAAgP8/wB+D64DTveSWR7IAAAATgAAA
AEHAAAAAKYAAAAAAAEAAgAC//wEZDAxMSRkMDNhMjQwZS04MWJkLTEx
ZTctODQxMC00MjAxMGE5NDAwMDKsXTyk
'/*!*/;
# at 1793
#170815 13:29:50 server id 200  end_log_pos 1824 CRC32
0x4f63aa2e  Xid = 59
COMMIT/*!*/;
```

The extracted log can be piped to MySQL to replay the events. It is better to use the force option while replaying binlogs because, if it is stuck at one point, it won't stop executing. Later, you can figure out the errors and fix the data manually.

```
shell> sudo mysqlbinlog /data/mysql/binlogs/server1.000001 |
mysql -f -h <remote_host> -u <username> -p
```

Or you can save into a file and execute later:

```
shell> sudo mysqlbinlog /data/mysql/binlogs/server1.000001 >
server1.binlog_extract
```

```
shell> cat server1.binlog_extract | mysql -h <remote_host> -
u <username> -p
```

Extracting based on time and position

You can extract the partial data from the binary log by specifying the position. Suppose you want to do point-in-time recovery. Assume a `DROP DATABASE` command was executed at `2017-08-19 12:18:00` and the latest available backup was `2017-08-19 12:00:00`, which you already restored. Now, you need to restore data from `12:00:01` till `2017-08-19 12:17:00`. Remember, if you extract the full log, it will also contain the `DROP DATABASE` command and it will wipe your data again.

You can extract the data by specifying the time window through `--start-datetime` and `--stop-datatime` options.

```
shell> sudo mysqlbinlog /data/mysql/binlogs/server1.000001 -
-start-datetime="2017-08-19 00:00:01"  --stop-
datetime="2017-08-19 12:17:00" > binlog_extract
```

The disadvantage of using a time window is you will miss the transactions that happened at the second the disaster occurred. To avoid this, you have to use the file offset of the event in the binary log file.

A consistent backup saves the binlog file offset until which it has already backed up. Once the backup is restored, you have to extract the binlogs from the offset provided by the backup. You will learn more about backups in the next chapter.

Assume that the backup has given an offset of `471` and the `DROP DATABASE` command was executed at an offset of `1793`. You can use `-`

`-start-position` and `--stop-position` options to extract a log between offsets:

```
shell> sudo mysqlbinlog /data/mysql/binlogs/server1.000001 -
-start-position=471  --stop-position=1793 > binlog_extract
```

Make sure that the `DROP DATABASE` command does not appear again in the extracted binlog.

Extracting based on the database

Using the `--database` option, you can filter events of a specific database. If you give this multiple times, only the last option will be considered. This works very well for row-based replication. But for statement-based replication and `MIXED`, this gives output only when the default database is selected.

The following command extracts events from the employees database:

```
shell> sudo mysqlbinlog /data/mysql/binlogs/server1.000001 -
-database=employees > binlog_extract
```

As explained in the MySQL 8 reference manual, suppose the binary log was created by executing these statements using statement-based-logging:

```
mysql>
INSERT INTO test.t1 (i) VALUES(100);
INSERT INTO db2.t2 (j)  VALUES(200);

USE test;
INSERT INTO test.t1 (i) VALUES(101);
INSERT INTO t1 (i) VALUES(102);
INSERT INTO db2.t2 (j) VALUES(201);

USE db2;
INSERT INTO test.t1 (i) VALUES(103);
INSERT INTO db2.t2 (j) VALUES(202);
INSERT INTO t2 (j) VALUES(203);
```

`mysqlbinlog --database=test` does not output the first two `INSERT` statements because there is no default database.

It outputs the three `INSERT` statements following `USE test`, but not the three `INSERT` statements following `USE db2`.

`mysqlbinlog --database=db2` does not output the first two `INSERT` statements because there is no default database.

It does not output the three `INSERT` statements following `USE` test, but does output the three `INSERT` statements following `USE db2`.

Extracting a row event display

In row-based replication, by default, binary format is displayed. To view the `ROW` information, you have to pass the `--verbose` or `-v` option to `mysqlbinlog`. The binary format of row events are shown as comments in the form of *pseudo-SQL* statements with lines beginning with `###`. You can see that a single `UPDATE` statement is rewritten as an `UPDATE` statement for each row:

```
shell>  mysqlbinlog /data/mysql/binlogs/server1.000001 --
start-position=660 --stop-position=1298 --verbose
~
~
# at 660
#170815 13:29:02 server id 200  end_log_pos 722 CRC32
0xe0a2ec74     Table_map: `employees`.`salaries` mapped to
number 165
# at 722
#170815 13:29:02 server id 200  end_log_pos 1298 CRC32
0xf0ef8b05     Update_rows: table id 165 flags: STMT_END_F

BINLOG '
HveSWRPIAAAAPgAAANICAAAAAKUAAAAAAAAEACWVtcGxveWVlcwAIc2FsYXJp
ZXMABAMDCgoAAAEB
AHTsouA=
~
~
'/*!*/;
### UPDATE `employees`.`salaries`
### WHERE
###   @1=10001
###   @2=240468
###   @3='1986:06:26'
###   @4='1987:06:26'
### SET
###   @1=10001
###   @2=120234
###   @3='1986:06:26'
###   @4='1987:06:26'
```

```
~
~
### UPDATE `employees`.`salaries`
### WHERE
###   @1=10001
###   @2=400000
###   @3='2017:06:18'
###   @4='9999:01:01'
### SET
###   @1=10001
###   @2=200000
###   @3='2017:06:18'
###   @4='9999:01:01'
SET @@SESSION.GTID_NEXT= 'AUTOMATIC' /* added by mysqlbinlog
*/ /*!*/;
DELIMITER ;
# End of log file
/*!50003 SET COMPLETION_TYPE=@OLD_COMPLETION_TYPE*/;
/*!50530 SET @@SESSION.PSEUDO_SLAVE_MODE=0*/;
```

If you want to just see the pseudo-SQL without the binary row information, specify `--base64-output="decode-rows"` along with `--verbose`:

```
shell>  sudo mysqlbinlog /data/mysql/binlogs/server1.000001
--start-position=660 --stop-position=1298 --verbose --
base64-output="decode-rows"
/*!50530 SET @@SESSION.PSEUDO_SLAVE_MODE=1*/;
/*!50003 SET
@OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
DELIMITER /*!*/;
# at 660
#170815 13:29:02 server id 200  end_log_pos 722 CRC32
0xe0a2ec74     Table_map: `employees`.`salaries` mapped to
number 165
# at 722
#170815 13:29:02 server id 200  end_log_pos 1298 CRC32
0xf0ef8b05     Update_rows: table id 165 flags: STMT_END_F
### UPDATE `employees`.`salaries`
### WHERE
###   @1=10001
###   @2=240468
###   @3='1986:06:26'
###   @4='1987:06:26'
### SET
###   @1=10001
###   @2=120234
###   @3='1986:06:26'
###   @4='1987:06:26'
~
```

Rewriting a database name

Suppose you want to restore the binary log of the `employees` database on a production server as `employees_dev` on a development server. You can use the `--rewrite-db='from_name->to_name'` option. This will rewrite all occurrences of `from_name` to `to_name`.

To convert multiple databases, specify the option multiple times:

```
shell> sudo mysqlbinlog /data/mysql/binlogs/server1.000001 -
-start-position=1499 --stop-position=1646 --rewrite-
db='employees->employees_dev'
~
# at 1499
#170815 13:29:27 server id 200  end_log_pos 1646 CRC32
0xc73d68fb     Query    thread_id=8    exec_time=0
error_code=0
use `employees_dev`/*!*/;
~
~
UPDATE salaries SET salary=salary*2 WHERE emp_no<10002
/*!*/;
SET @@SESSION.GTID_NEXT= 'AUTOMATIC' /* added by mysqlbinlog
*/ /*!*/;
DELIMITER ;
# End of log file
~
```

You can see that the statement `` use `employees_dev`/*!*/; `` is used. So, while restoring, all the changes will be applied to `employees_dev database`.

As explained in the MySQL reference manual:

> *When used together with the* `--database` *option, the* `--rewrite-db` *option is applied first, then the* `--database`

option is applied using the rewritten database name. The order in which the options are provided makes no difference in this regard. This means that, for example, if `mysqlbinlog` *is started with* `--rewrite-db='mydb->yourdb' --database=yourdb`*, then all updates to any tables in databases* `mydb` *and* `yourdb` *are included in the output.*
On the other hand, if it is started with `--rewrite-db='mydb->yourdb' --database=mydb`*, then* `mysqlbinlog` *outputs no statements at all: since all updates to* `mydb` *are first rewritten as updates to* `yourdb` *before applying the* `--database` *option, there remain no updates that match* `--database=mydb`*.*

Disabling a binary log for recovery

While restoring binary logs, if you don’t want the `mysqlbinlog` process to create binary logs, you can use the `--disable-log-bin` option so that binary logs won't be written:

```
shell> sudo mysqlbinlog /data/mysql/binlogs/server1.000001 -
-start-position=660 --stop-position=1298 --disable-log-bin >
binlog_restore
```

You can see that `SQL_LOG_BIN=0` is written to the `binlog` restore file, which will prevent creating the binlogs.

```
/*!32316 SET @OLD_SQL_LOG_BIN=@@SQL_LOG_BIN, SQL_LOG_BIN=0*/;
```

Displaying events in a binary log file

Apart from using `mysqlbinlog`, you can also use the `SHOW BINLOG EVENTS` command to display the events.

The following command will display the events in the `server1.000008` binary log. If `LIMIT` is not specified, all the events are displayed:

```
mysql> SHOW BINLOG EVENTS IN 'server1.000008' LIMIT 10;
+----------------+-----+----------------+-----------+-------
------+-----------------------------------------+
| Log_name       | Pos | Event_type     | Server_id |
End_log_pos | Info                                    |
+----------------+-----+----------------+-----------+-------
------+-----------------------------------------+
| server1.000008 |   4 | Format_desc    |       200 |
123 | Server ver: 8.0.3-rc-log, Binlog ver: 4 |
| server1.000008 | 123 | Previous_gtids |       200 |
154 |                                         |
| server1.000008 | 154 | Anonymous_Gtid |       200 |
226 | SET @@SESSION.GTID_NEXT= 'ANONYMOUS'     |
| server1.000008 | 226 | Query          |       200 |
336 | drop database company /* xid=4134 */     |
| server1.000008 | 336 | Anonymous_Gtid |       200 |
408 | SET @@SESSION.GTID_NEXT= 'ANONYMOUS'     |
| server1.000008 | 408 | Query          |       200 |
485 | BEGIN                                    |
| server1.000008 | 485 | Table_map      |       200 |
549 | table_id: 975 (employees.emp_details)    |
| server1.000008 | 549 | Write_rows     |       200 |
804 | table_id: 975 flags: STMT_END_F          |
| server1.000008 | 804 | Xid            |       200 |
835 | COMMIT /* xid=9751 */                    |
| server1.000008 | 835 | Anonymous_Gtid |       200 |
907 | SET @@SESSION.GTID_NEXT= 'ANONYMOUS'     |
+----------------+-----+----------------+-----------+-------
```

```
------+--------------------------------------+
10 rows in set (0.00 sec)
```

You can also specify the position and offset:

```
mysql> SHOW BINLOG EVENTS IN 'server1.000008' FROM 123 LIMIT
2,1;
+---------------+-----+------------+-----------+----------
--+-------------------------------------+
| Log_name      | Pos | Event_type | Server_id |
End_log_pos | Info                                 |
+---------------+-----+------------+-----------+----------
--+-------------------------------------+
| server1.000008 | 226 | Query      |       200 |
336 | drop database company /* xid=4134 */ |
+---------------+-----+------------+-----------+----------
--+-------------------------------------+
1 row in set (0.00 sec)
```

Ignoring databases to write to a binary log

You can choose which databases should be written to the binary log by specifying the `--binlog-do-db=db_name` option in `my.cnf`. To specify multiple databases you *must* use multiple instances of this option. Because database names can contain commas, the list will be treated as the name of a single database if you supply a comma-separated list. You need to restart the MySQL server to effect changes.

How to do it...

Open `my.cnf` and add the following lines:

```
shell> sudo vi /etc/my.cnf
[mysqld]
binlog_do_db=db1
binlog_do_db=db2
```

The behavior on `binlog-do-db` changes from statement-based logging to row-based logging just like the `--database` option in the `mysqlbinlog` utility.

In statement-based logging, only those statements are written to the binary log where the default database (that is, the one selected by `USE`) is written to the binary log. You should be very careful while using the `binlog-do-db` option because it does not work as you might expect when using statement-based logging. Go through the following examples mentioned in the reference manual.

Example 1

If the server is started with `--binlog-do-db=sales` and you issue the following statements, the `UPDATE` statement is not logged:

```
mysql> USE prices;
mysql> UPDATE sales.january SET amount=amount+1000;
```

The main reason for this *just check the default database* behavior is that it is difficult from the statement alone to know whether it should be replicated. It is also faster to check only the default database rather than all databases if there is no need.

Example 2

If the server is started with `--binlog-do-db=sales`, the following `UPDATE` statement is logged even though prices were not included when setting `--binlog-do-db`:

```
mysql> USE sales;
mysql> UPDATE prices.discounts SET percentage = percentage +
10;
```

Because sales is the default database when the `UPDATE` statement is issued, the `UPDATE` is logged.

In row-based logging, it is restricted to database `db_name`. Only changes to tables belonging to `db_name` are logged; the default database has no effect on this.

Another important difference in `--binlog-do-db` handling for statement-based logging, as opposed to the row-based logging, occurs with regard to statements that refer to multiple databases. Suppose that the server is started with `--binlog-do-db=db1`, and the following statements are executed:

```
mysql> USE db1;
mysql> UPDATE db1.table1 SET col1 = 10, db2.table2 SET col2
= 20;
```

If you are using statement-based logging, the updates to both tables are written to the binary log. However, when using the row-based format, only the changes to `table1` are logged; `table2` is in a different database, so it is not changed by the `UPDATE`.

Similarly, you can use the `--binlog-ignore-db=db_name` option to ignore the database from writing to the binary log.

Refer to the manual for more information, at: https://dev.mysql.com/doc/refman/8.0/en/replication-rules.html.

Relocating binary logs

Since binary logs occupy more space, and sometimes you may wish to change the location of binary logs, the following procedure helps. Changing the `log_bin` alone is not sufficient, you have to move all the binary logs and update the index file with the new location. The `mysqlbinlogmove` utility eases your work by automating those tasks.

How to do it...

MySQL Utilities should be installed for using the `mysqlbinlogmove` script. Refer to Chapter 1, *MySQL 8.0 – Installing and Upgrading*, for installation steps.

1. Stop the MySQL server:

```
shell> sudo systemctl stop mysql
```

2. Start the `mysqlbinlogmove` utility. If you want to change the binary logs from `/data/mysql/binlogs` to `/binlogs`, the following command should be used. If your base name is not default, you have to mention your base name through the `--bin-log-base name` option:

```
shell> sudo mysqlbinlogmove --bin-log-base name=server1
--binlog-dir=/data/mysql/binlogs /binlogs
#
# Moving bin-log files...
# - server1.000001
# - server1.000002
# - server1.000003
# - server1.000004
# - server1.000005
#
#...done.
#
```

3. Edit the `my.cnf` file and update the new location of `log_bin`:

```
shell> sudo vi /etc/my.cnf
[mysqld]
log_bin=/binlogs
```

4. Start the MySQL server:

```
      shell> sudo systemctl start mysql
```

The new location is updated in AppArmor or SELinux.

If there are a lot of binlogs, the downtime of the server will be high. To avoid that you can use the `--server` option to relocate all binary logs except the ones currently in use (with the higher sequence number). Then stop the server, use the preceding method, and relocate the last binary log, which will be much faster because only one file is there. Then you can change the `my.cnf` and start the server.

For example:

```
shell> sudo mysqlbinlogmove --server=root:pass@host1:3306
/new/location
```

Backups

In this chapter, we will cover the following recipes:

- Taking backups using mysqldump
- Taking backups using mysqlpump
- Taking backups using mydumper
- Taking backups using flat files
- Taking backups using XtraBackup
- Locking instances for backup
- Binary log backup

Introduction

After setting up the database, the next important thing is to set up backups. In this chapter, you will learn how to set up various types of backups. There are two main ways to perform a backup. One is logical backup, which exports all the databases, table structures, data, and stored routines into a set of SQL statements that can be executed again to recreate the state of the database. The other type is physical backup, which contains all the files on the system that the databases used to store all the database entities:

- **Logical backup utilities**: `mysqldump`, `mysqlpump`, and `mydumper` (not shipped with MySQL)
- **Physical backup utilities**: XtraBackup (not shipped with MySQL) and flat file backup

For point-in-time recovery, the backup should be able to provide the binary log positions up to which the backups are taken. This is called a **consistent backup**.

It is highly recommended to take backups from a slave onto a filer mounted on it.

Taking backups using mysqldump

`mysqldump` is a widely used logical backup tool. It gives a variety of options to include or exclude databases, select specific data to be backed up, back up only the schema without data, or just take a backup of stored routines without anything else, and more.

How to do it...

The `mysqldump` utility comes along with the `mysql` binary, so you do need to install it separately. Most production scenarios are covered in this section.

The syntax is as follows:

```
shell> mysqldump [options]
```

In the options, you can specify the username, password, and hostname to connect to the database, like this:

```
--user <user_name> --password <password>
or
-u <user_name> -p<password>
```

In this chapter, `--user` and `--password` are not mentioned in every example, to keep the reader focused on other important options.

Full backup of all databases

This can be done with the following:

```
shell> mysqldump --all-databases > dump.sql
```

The `--all-databases` option takes a backup of all databases and all tables. The `>` operator redirects the output to the `dump.sql` file. Prior to MySQL 8, stored procedures and events were stored in the `mysql.proc` and `mysql.event` tables. From MySQL 8 onward, definitions for corresponding objects are stored in `data dictionary` tables, but those tables are not dumped. To include stored routines and events in a dump made using `--all-databases`, use the `--routines` and `--events` options.

To include the routines and events:

```
shell> mysqldump --all-databases --routines --events >
dump.sql
```

You can open the `dump.sql` file to see how it is structured. The first few lines are the session variables at the time of dumping. Next is the `CREATE DATABASE` statement, followed by the `USE DATABASE` command. Next is the `DROP TABLE IF EXISTS` statement, followed by `CREATE TABLE`; and then we have the actual `INSERT` statements that insert the data. Since the data is stored as SQL statements, this is called **logical backup**.

You will notice that when you restore the dump, the `DROP TABLE` statement will wipe off all the tables before creating the tables.

Point-in-time recovery

To get point-in-time recovery, you should specify `--single-transaction` and `--master-data`.

The `--single-transaction` option provides consistent backup by changing the transaction isolation mode to `REPEATABLE READ` and executing `START TRANSACTION` before taking a backup. It is useful only with transactional tables, such as `InnoDB`, because then it dumps the consistent state of the database at the time `START TRANSACTION` was issued without blocking any applications.

The `--master-data` option prints the binary log coordinates of the server to the `dump` file. If `--master-data=2`, it prints as a comment. This also uses the `FLUSH TABLES WITH READ LOCK` statement to get a snapshot for the binary logs. As explained in Chapter 5, *Transactions*, this can be very dangerous when there is any long-running transaction:

```
shell> mysqldump --all-databases --routines --events --
single-transaction --master-data > dump.sql
```

Dumping master binary coordinates

Backups are always taken on slaves. To get the binary log coordinates of the master when the backup was taken, you can use the `--dump-slave` option. If you are taking the binary log backup from the master, use this option. Otherwise, use the `--master-data` option:

```
shell> mysqldump --all-databases --routines --events --
single-transaction --dump-slave > dump.sql
```

The output will be like this:

```
--
-- Position to start replication or point-in-time recovery
from (the master of this slave)
--
CHANGE MASTER TO MASTER_LOG_FILE='centos7-bin.000001',
MASTER_LOG_POS=463;
```

Specific databases and tables

To back up only a specific database, execute this:

```
shell> mysqldump --databases employees >
employees_backup.sql
```

To back up only a specific table, execute this:

```
shell> mysqldump --databases employees --tables employees >
employees_backup.sql
```

Ignore tables

To ignore certain tables, you can use the `--ignore-table=database.table` option. To specify more than one table to ignore, use the directive multiple times:

```
shell> mysqldump --databases employees --ignore-
table=employees.salary > employees_backup.sql
```

Specific rows

`mysqldump` helps you filter the data you back up. Suppose you want to take a backup of employees who joined after 2000:

```
shell> mysqldump --databases employees --tables employees --
databases employees --tables employees  --
where="hire_date>'2000-01-01'" > employees_after_2000.sql
```

You can use the `LIMIT` clause to limit the results:

```
shell> mysqldump --databases employees --tables employees --
databases employees --tables employees  --where="hire_date
>= '2000-01-01' LIMIT 10" >
employees_after_2000_limit_10.sql
```

Backup from a remote server

Sometimes, you may not have SSH access to the database server (as in the case of cloud instances such as Amazon RDS). In those cases, you can use `mysqldump` to take a backup from a remote server to the local server. For this, you need to mention `hostname` using the `--hostname` option. Make sure that the user has appropriate privileges to connect and perform a backup:

```
shell> mysqldump --all-databases --routines --events --
triggers --hostname <remote_hostname> > dump.sql
```

Backup to rebuild another server with a different schema

There can be a situation where you want to have a different schema on another server. In that case, you have to dump and restore the schema, alter the schema as you need, and then dump and restore the data. Altering the schema with data can take a long time, depending on the amount of data you have. Note that this method will work only when the modified schema is compatible with inserts. The modified table can have extra columns but it should have all the columns present in the original table.

Only schema and no data

You can use `--no-data` to dump only the schema:

```
shell> mysqldump --all-databases --routines --events --
triggers --no-data > schema.sql
```

Only data and no schema

You can use the following options to take only a data dump, excluding the schema.

`--complete-insert` will print the column names in the `INSERT` statement, which will help when you have extra columns in the modified table:

```
shell> mysqldump --all-databases --no-create-db --no-create-
info --complete-insert > data.sql
```

Backup for merging data with other server

You can take backup either way to replace the old data or keep the old data incase of conflict.

REPLACE with new data

Suppose you want to restore data from the production database to a development machine that already has some data. If you want to merge the data from production with development, you can use the `--replace` option, which will use the `REPLACE INTO` statement instead of the `INSERT` statement. You should also include the `--skip-add-drop-table` option, which will not write a `DROP TABLE` statement to the `dump` file. If you have the same number of tables and structure, you can also include the `--no-create-info` option, which will skip the `CREATE TABLE` statement in the `dump` file:

```
shell> mysqldump --databases employees --skip-add-drop-table
--no-create-info --replace > to_development.sql
```

If you have some extra tables in production, the preceding dump will fail while restoring because the table does not exist on development server. In that case, you should not add the `--no-create-info` option and use the `force` option while restoring. Otherwise, the restore will fail at `CREATE TABLE` saying that the table already exists. Unfortunately, `mysqldump` has not provided the option of `CREATE TABLE IF NOT EXISTS`.

IGNORE data

Instead of `REPLACE`, you can use the `INSERT IGNORE` statement when writing to the `dump` file. This will keep the existing data on the server and insert new data.

Taking backups using mysqlpump

`mysqlpump` is a very similar program to `mysqldump` with some extra features.

How to do it...

There are numerous ways to do it. Let's have a look at each one in detail.

Parallel processing

You can speed up the process of dumping by specifying the number of threads (based on the number of CPUs). For example, use eight threads to take a full backup:

```
shell> mysqlpump --default-parallelism=8 > full_backup.sql
```

You can even specify the number of threads for each database. In our case, the `employees` database is very big compared to the `company` database. So you can spawn four threads to `employees` and two threads to the `company` database:

```
shell> mysqlpump -u root --password --parallel-
schemas=4:employees --default-parallelism=2 >
full_backup.sql
Dump progress: 0/6 tables, 250/331145 rows
Dump progress: 0/34 tables, 494484/3954504 rows
Dump progress: 0/42 tables, 1035414/3954504 rows
Dump progress: 0/45 tables, 1586055/3958016 rows
Dump progress: 0/45 tables, 2208364/3958016 rows
Dump progress: 0/45 tables, 2846864/3958016 rows
Dump progress: 0/45 tables, 3594614/3958016 rows
Dump completed in 6957
```

Another example to distribute the threads has three threads for `db1` and `db2`, two threads for `db3` and `db4`, and four threads for the rest of the databases:

```
shell> mysqlpump --parallel-schemas=3:db1,db2 --parallel-
schemas=2:db3,db4 --default-parallelism=4 > full_backup.sql
```

You will notice that there is a progress bar that will help you estimate the time.

Exclude/include database objects using regex

Take a backup of all databases ending with `prod`:

```
shell> mysqlpump --include-databases=%prod --result-
file=db_prod.sql
```

Suppose there are some test tables in some databases and you want to exclude them from the backup; you can specify using the `--exclude-tables` option, which will exclude tables with the name `test` across all databases:

```
shell> mysqlpump --exclude-tables=test --result-
file=backup_excluding_test.sql
```

The value of each inclusion and exclusion option is a comma-separated list of names of the appropriate object type. Wildcard characters are permitted in object names:

- `%` matches any sequence of zero or more characters
- `_` matches any single character

Apart from databases and tables, you can also include or exclude triggers, routines, events, and users, for example, `--include-routines`, `--include-events`, and `--exclude-triggers`.

To know more about the include and exclude options, refer to https://dev.mysql.com/doc/refman/8.0/en/mysqlpump.html#mysqlpump-filtering.

Backup users

In `mysqldump`, you will not get a backup of the users in `CREATE USER` or `GRANT` statements; instead you have to take a backup of the `mysql.user` table. Using `mysqlpump`, you can dump user accounts as account management statements (`CREATE USER` and `GRANT`) rather than as inserts into the `mysql` system database:

```
shell> mysqlpump --exclude-databases=% --users >
users_backup.sql
```

You can also exclude some users by specifying the `--exclude-users` option:

```
shell> mysqlpump --exclude-databases=% --exclude-users=root
--users > users_backup.sql
```

Compressed backups

You can compress the backup to minimize disk space and network bandwidth. You can use `--compress-output=lz4` or `--compress-output=zlib`.

Note that you should have the appropriate decompress utilities:

```
shell> mysqlpump -u root -pxxxx --compress-output=lz4 >
dump.lz4
```

To decompress execute this:

```
shell> lz4_decompress dump.lz4 dump.sql
```

Using `zlib` execute this:

```
shell> mysqlpump -u root -pxxxx --compress-output=zlib >
dump.zlib
```

To decompress execute this:

```
shell> zlib_decompress dump.zlib dump.sql
```

Faster reload

You will notice that in the output, the secondary indexes are omitted from the `CREATE TABLE` statement. This will speed up the restoration process. The indexes are added at the end of the `INSERT` using the `ALTER TABLE` statement.

Indexes will be covered in Chapter 13, *Performance Tuning*.

Previously, it was possible to dump all tables in the `mysql` *system database. As of MySQL 8,* `mysqldump` *and* `mysqlpump` *dump only non-*`data dictionary` *tables in that database.*

Taking backups using mydumper

`mydumper` is a logical backup tool that's similar to `mysqlpump`.

`mydumper` has these advantages over `mysqldump`:

- Parallelism (hence, speed) and performance (avoids expensive character set conversion routines and has efficient code overall).
- Consistency. It maintains snapshots across all threads, provides accurate master and slave log positions, and so on. `mysqlpump` does not guarantee consistency.
- Easier to manage output (separate files for tables and dumped metadata, and it is easy to view/parse data). `mysqlpump` writes everything to one file, which limits the option of loading selective database objects.
- Inclusion and exclusion of database objects using regex.
- The option to kill long-running transactions that block the backup and all subsequent queries.

`mydumper` is an open source backup tool, which you need to install separately. In this section, installation steps on Debian and Red Hat systems and the usage of `mydumper` will be covered.

How to do it...

Let's start with the installation and then we will learn a lot of things related to backup in every subsection that is listed within this recipe.

Installation

Install the prerequisites:

On Ubuntu/Debain:

```
shell> sudo apt-get install libglib2.0-dev libmysqlclient-
dev zlib1g-dev libpcre3-dev cmake git
```

On Red Hat/CentOS/Fedora:

```
shell> yum install glib2-devel mysql-devel zlib-devel pcre-
devel cmake gcc-c++ git
shell> cd /opt
shell> git clone https://github.com/maxbube/mydumper.git
shell> cd mydumper
shell> cmake .

shell> make
Scanning dependencies of target mydumper
[ 25%] Building C object
CMakeFiles/mydumper.dir/mydumper.c.o
[ 50%] Building C object
CMakeFiles/mydumper.dir/server_detect.c.o
[ 75%] Building C object
CMakeFiles/mydumper.dir/g_unix_signal.c.o

shell> make install
[ 75%] Built target mydumper
[100%] Built target myloader
Linking C executable CMakeFiles/CMakeRelink.dir/mydumper
Linking C executable CMakeFiles/CMakeRelink.dir/myloader
Install the project...
-- Install configuration: ""
-- Installing: /usr/local/bin/mydumper
-- Installing: /usr/local/bin/myloader
```

Alternatively, using YUM or APT, you can find the releases here at https://github.com/maxbube/mydumper/releases:

```
#YUM
shell> sudo yum install -y
"https://github.com/maxbube/mydumper/releases/download/v0.9.
3/mydumper-0.9.3-41.el7.x86_64.rpm"

#APT
shell> wget
"https://github.com/maxbube/mydumper/releases/download/v0.9.
3/mydumper_0.9.3-41.jessie_amd64.deb"

shell> sudo dpkg -i mydumper_0.9.3-41.jessie_amd64.deb
shell> sudo apt-get install -f
```

Full backup

The following command takes a backup of all databases into the `/backups` folder:

```
shell> mydumper -u root --password=<password> --outputdir
/backups
```

Several files are created in the `/backups` folder. Each database has its `CREATE DATABASE` statement as `<database_name>-schema-create.sql` and each table will have its own schema and data files. Schema files are stored as `<database_name>.<table>-schema.sql` and data files are stored as `<database_name>.<table>.sql`.

The views are stored as `<database_name>.<table>-schema-view.sql`. Stored routines, triggers, and events are stored as `<database_name>-schema-post.sql` (use `sudo mkdir –pv /backups` if directory is not created):

```
shell> ls -lhtr /backups/company*
-rw-r--r-- 1 root root 69 Aug 13 10:11 /backups/company-
schema-create.sql
-rw-r--r-- 1 root root 180 Aug 13 10:11
/backups/company.payments.sql
-rw-r--r-- 1 root root 239 Aug 13 10:11
/backups/company.new_customers.sql
-rw-r--r-- 1 root root 238 Aug 13 10:11
/backups/company.payments-schema.sql
-rw-r--r-- 1 root root 303 Aug 13 10:11
/backups/company.new_customers-schema.sql
-rw-r--r-- 1 root root 324 Aug 13 10:11
/backups/company.customers-schema.sql
```

If there are any queries longer than 60 seconds, `mydumper` will fail with the following error:

```
** (mydumper:18754): CRITICAL **: There are queries in
PROCESSLIST running longer than 60s, aborting dump,
 use --long-query-guard to change the guard value, kill
queries (--kill-long-queries) or use  different server for
dump
```

To avoid this, you can pass the `--kill-long-queries` option or set `--long-query-guard` to a higher value.

The `--kill-long-queries` option kills all the queries that are greater than 60 seconds or a value set by `--long-query-guard`. Please note that `--kill-long-queries` also kills the replication thread due to a bug (https://bugs.launchpad.net/mydumper/+bug/1713201):

```
shell> sudo mydumper --kill-long-queries --outputdir
/backups
** (mydumper:18915): WARNING **: Using trx_consistency_only,
binlog coordinates will not be accurate if you are writing
to non transactional tables.
** (mydumper:18915): WARNING **: Killed a query that was
running for 368s
```

Consistent backup

The metadata file in the `backup` directory contains the binary log coordinates for consistent backup.

On a master, it captures the binary log positions:

```
shell> sudo cat /backups/metadata
Started dump at: 2017-08-20 12:44:09
SHOW MASTER STATUS:
    Log: server1.000008
    Pos: 154
    GTID:
```

On a slave, it captures the binary log positions of both master and slave:

```
shell> cat /backups/metadata
Started dump at: 2017-08-26 06:26:19
SHOW MASTER STATUS:
 Log: server1.000012
 Pos: 154
 GTID:
SHOW SLAVE STATUS:
 Host: 35.186.158.188
 Log: master-bin.000013
```

```
 Pos: 4633
 GTID:
Finished dump at: 2017-08-26 06:26:24
```

Backup of a single table

The following command takes the backup of the `employees` table of the `employees` database into the `/backups` directory:

```
shell> mydumper -u root --password=<password> -B employees -
T employees --triggers --events --routines  --outputdir
/backups/employee_table
```

```
shell> ls -lhtr /backups/employee_table/
total 17M
-rw-r--r-- 1 root root 71 Aug 13 10:35 employees-schema-
create.sql
-rw-r--r-- 1 root root 397 Aug 13 10:35 employees.employees-
schema.sql
-rw-r--r-- 1 root root 3.4K Aug 13 10:35 employees-schema-
post.sql
-rw-r--r-- 1 root root 75 Aug 13 10:35 metadata
-rw-r--r-- 1 root root 17M Aug 13 10:35
employees.employees.sql
```

The convention of the files are as follows:

- `employees-schema-create.sql` contains the `CREATE DATABASE` statement
- `employees.employees-schema.sql` contains the `CREATE TABLE` statement
- `employees-schema-post.sql` contains the `ROUTINES`, `TRIGGERS`, and `EVENTS`
- `employees.employees.sql` contains the actual data in the form of `INSERT` statements

Backup of specific databases using regex

You can include/exclude specific databases using the `regex` option. The following command will exclude `mysql` and `test` databases from the backup:

```
shell> mydumper -u root --password=<password> --regex '^(?!
(mysql|test))' --outputdir /backups/specific_dbs
```

Taking backup of a big table using mydumper

To speed up the dump and restore of a big table, you can split it into small chunks. The chunk size can be specified by the number of rows it contains and each chunk will be written into a separate file:

```
shell> mydumper -u root --password=<password> -B employees -
T employees --triggers --events --routines --rows=10000 -t 8
--trx-consistency-only --outputdir
/backups/employee_table_chunks
```

- `-t`: Specifies the number of threads
- `--trx-consistency-only`: If you are using only transnational tables, such as `InnoDB`, using this option will minimize the locking
- `--rows`: Split the table into chunks of this number of rows

For each chunk, a file is created as `<database_name>.<table_name>.<number>.sql`; the number is padded with five zeros:

```
shell> ls -lhr /backups/employee_table_chunks
total 17M
-rw-r--r-- 1 root root 71 Aug 13 10:45 employees-schema-
create.sql
-rw-r--r-- 1 root root 75 Aug 13 10:45 metadata
-rw-r--r-- 1 root root 397 Aug 13 10:45 employees.employees-
schema.sql
-rw-r--r-- 1 root root 3.4K Aug 13 10:45 employees-schema-
post.sql
-rw-r--r-- 1 root root 633K Aug 13 10:45
employees.employees.00008.sql
-rw-r--r-- 1 root root 634K Aug 13 10:45
employees.employees.00002.sql
```

```
-rw-r--r-- 1 root root 1.3M Aug 13 10:45
employees.employees.00006.sql
-rw-r--r-- 1 root root 1.9M Aug 13 10:45
employees.employees.00004.sql
-rw-r--r-- 1 root root 2.5M Aug 13 10:45
employees.employees.00000.sql
-rw-r--r-- 1 root root 2.5M Aug 13 10:45
employees.employees.00001.sql
-rw-r--r-- 1 root root 2.6M Aug 13 10:45
employees.employees.00005.sql
-rw-r--r-- 1 root root 2.6M Aug 13 10:45
employees.employees.00009.sql
-rw-r--r-- 1 root root 2.6M Aug 13 10:45
employees.employees.00010.sql
```

Non-blocking backup

To provide consistent backup, `mydumper` acquires `GLOBAL LOCK` by executing `FLUSH TABLES WITH READ LOCK`.

You have already seen how dangerous it is to use `FLUSH TABLES WITH READ LOCK` if there are any long-running transactions (explained in Chapter 5, *Transactions*). To avoid that, you can pass the `--kill-long-queries` option to kill blocking queries rather than aborting `mydumper`.

- `--trx-consistency-only`: This is equivalent to `--single-transaction` for `mysqldump` but with a `binlog` position. Obviously, this position only applies to transactional tables. One of the advantages of using this option is that the global read lock is only held for the threads' coordination, so it's released as soon as the transactions are started.
- `--use-savepoints` reduces metadata locking issues (needs the `SUPER` privilege).

Compressed backups

You can specify the `--compress` option to compress the backup:

```
shell> mydumper -u root --password=<password> -B employees -
T employees -t 8 --trx-consistency-only --compress --
outputdir /backups/employees_compress
```

```
shell> ls -lhtr /backups/employees_compress
total 5.3M
-rw-r--r-- 1 root root 91 Aug 13 11:01 employees-schema-
create.sql.gz
-rw-r--r-- 1 root root 263 Aug 13 11:01 employees.employees-
schema.sql.gz
-rw-r--r-- 1 root root 75 Aug 13 11:01 metadata
-rw-r--r-- 1 root root 5.3M Aug 13 11:01
employees.employees.sql.gz
```

Backing up only data

You can use the `--no-schemas` option to skip the schema and take a data-only backup:

```
shell> mydumper -u root --password=<password> -B employees -
T employees -t 8 --no-schemas --compress --trx-consistency-
only --outputdir /backups/employees_data
```

Taking backups using flat files

This is a physical backup method whereby you take a backup by directly copying the files inside `data directory`. Since new data is written while you copy the files, the backup will be inconsistent and cannot be used. To avoid that, you have to shut down MySQL, copy the files, and then start MySQL. This method is not used for daily backups but is well suited during maintenance windows for upgrades or downgrades or while doing a host swap.

How to do it...

1. Shut down the MySQL server:

```
shell> sudo service mysqld stop
```

2. Copy the files into the `data directory` (your directory may be different):

```
shell> sudo rsync -av /data/mysql /backups
or do rsync over ssh to remote server
shell> rsync -e ssh -az /data/mysql/
backup_user@remote_server:/backups
```

3. Start the MySQL server:

```
shell> sudo service mysqld start
```

Taking backups using XtraBackup

XtraBackup is an open source backup software provided by Percona. It copies flat files without shutting down the server, but to avoid inconsistencies, it uses a redo log file. It is widely used by many companies as a standard backup tool. The advantages are that it is very fast compared to logical backup tools and recovery is also very fast.

This is how Percona XtraBackup works (taken from the Percona XtraBackup documentation):

1. It copies your `InnoDB` data files, which results in data that is internally inconsistent; but then it performs crash recovery on the files to make them a consistent, usable database again.
2. This works because `InnoDB` maintains a redo log, also called the transaction log. This contains a record of every change to the `InnoDB` data. When `InnoDB` starts, it inspects the data files and the transaction log, and performs two steps. It applies committed transaction log entries to the data files, and it performs an undo operation on any transactions that modified data but did not commit.

3. Percona XtraBackup works by remembering the **log sequence number** (**LSN**) when it starts, and then copying away the data files. It takes some time to do this, so if the files are changing, then they reflect the state of the database at different points in time. At the same time, Percona XtraBackup runs a background process that watches the transaction log files, and copies changes from it. Percona XtraBackup needs to do this

continually because the transaction logs are written in a round-robin fashion, and can be reused after a while. Percona XtraBackup needs the transaction log records for every change to the data files since it began execution.

How to do it...

At the time of writing, Percona XtraBackup is not supported for MySQL 8. Eventually, Percona will release a new version of XtraBackup supporting MySQL 8; hence only the installation is covered.

Installation

The installation steps are in the following sections.

On CentOS/Red Hat/Fedora

1. Install `mysql-community-libs-compat`:

   ```
   shell> sudo yum install -y mysql-community-libs-compat
   ```

2. Install the Percona repository:

   ```
   shell> sudo yum install
   http://www.percona.com/downloads/percona-
   release/redhat/0.1-4/percona-release-0.1-4.noarch.rpm
   ```

   You should see some output such as the following:

   ```
   Retrieving http://www.percona.com/downloads/percona-
   release/redhat/0.1-4/percona-release-0.1-4.noarch.rpm
   Preparing...
   ########################################### [100%]
      1:percona-release
   ########################################### [100%]
   ```

3. Test the repository:

   ```
   shell> yum list | grep xtrabackup
   holland-xtrabackup.noarch 1.0.14-3.el7 epel
   percona-xtrabackup.x86_64 2.3.9-1.el7 percona-release-
   x86_64
   percona-xtrabackup-22.x86_64 2.2.13-1.el7 percona-
   release-x86_64
   percona-xtrabackup-22-debuginfo.x86_64 2.2.13-1.el7
   percona-release-x86_64
   percona-xtrabackup-24.x86_64 2.4.8-1.el7 percona-
   release-x86_64
   percona-xtrabackup-24-debuginfo.x86_64 2.4.8-1.el7
   percona-release-x86_64
   percona-xtrabackup-debuginfo.x86_64 2.3.9-1.el7
   percona-release-x86_64
   percona-xtrabackup-test.x86_64 2.3.9-1.el7 percona-
   release-x86_64
   ```

```
percona-xtrabackup-test-22.x86_64 2.2.13-1.el7 percona-
release-x86_64
percona-xtrabackup-test-24.x86_64 2.4.8-1.el7 percona-
release-x86_64
```

4. Install XtraBackup:

```
shell> sudo yum install percona-xtrabackup-24
```

On Debian/Ubuntu

1. Fetch the repository packages from Percona:

```
shell> wget https://repo.percona.com/apt/percona-
release_0.1-4.$(lsb_release -sc)_all.deb
```

2. Install the downloaded package with `dpkg`. To do that, run the following commands as `root` or with `sudo`:

```
shell> sudo dpkg -i percona-release_0.1-4.$(lsb_release
-sc)_all.deb
```

   Once you install this package, the Percona repositories should be added. You can check the repository setup in the `/etc/apt/sources.list.d/percona-release.list` file.

3. Remember to update the local cache:

```
shell> sudo apt-get update
```

4. After that, you can install the package:

```
shell> sudo apt-get install percona-xtrabackup-24
```

Locking instances for backup

As of MySQL 8, you can lock the instance for backup, which will allow the DML during online backup and block all the operations that could result in an inconsistent snapshot.

How to do it...

Before you begin your backup, lock the instance for backup:

```
mysql> LOCK INSTANCE FOR BACKUP;
```

Perform the backup, and after completion, unlock the instance:

```
mysql> UNLOCK INSTANCE;
```

Binary log backup

You know that binary logs are needed for point-in-time recovery. In this section, you will understand how to take a backup of binary logs. The process streams the binary logs from the database server to a remote backup server. You can take the binary log backup from either the slave or the master. If you are taking the binary log backup from the master and the actual backup from the slave, you should use `--dump-slave` to get the corresponding master log position. If you are using `mydumper` or XtraBackup, it gives both the master and slave binary log positions.

How to do it...

1. Create a replication user on the server. Create a strong password:

```
mysql> GRANT REPLICATION SLAVE ON *.* TO
'binlog_user'@'%' IDENTIFIED BY 'binlog_pass';
Query OK, 0 rows affected, 1 warning (0.03 sec)
```

2. Check the binary logs on the server:

```
mysql> SHOW BINARY LOGS;
+----------------+-----------+
| Log_name       | File_size |
+----------------+-----------+
| server1.000008 |      2451 |
| server1.000009 |       199 |
| server1.000010 |      1120 |
| server1.000011 |       471 |
| server1.000012 |       154 |
+----------------+-----------+
5 rows in set (0.00 sec)
```

   You can find the first binary log available on the server; from this, you can start the backup. In this case, it is `server1.000008`.

3. Log in to the backup server and execute the following command. This will copy the binary logs from the MySQL server to the backup server. You can start using `nohup` or `disown`:

```
shell> mysqlbinlog -u <user> -p<pass> -h <server> --
read-from-remote-server --stop-never
--to-last-log --raw server1.000008 &
shell> disown -a
```

4. Verify that the binary logs are being backed up:

```
shell> ls -lhtr server1.0000*
-rw-r-----. 1 mysql mysql 2.4K Aug 25 12:22
server1.000008
-rw-r-----. 1 mysql mysql  199 Aug 25 12:22
server1.000009
-rw-r-----. 1 mysql mysql 1.1K Aug 25 12:22
server1.000010
-rw-r-----. 1 mysql mysql  471 Aug 25 12:22
server1.000011
-rw-r-----. 1 mysql mysql  154 Aug 25 12:22
server1.000012
```

Restoring Data

In this chapter, we will cover the following recipes:

- Recovering from mysqldump and mysqlpump
- Recovering from mydumper using myloader
- Recovering from flat file backup
- Performing point-in-time recovery

Introduction

In this chapter, you will learn about various backup
restoration methods. Assume that the backups and binary logs are available on the server.

Recovering from mysqldump and mysqlpump

The logical backup tools `mysqldump` and `mysqlpump` write data to a single file.

How to do it...

Log in to the server where the backups are available:

```
shell> cat /backups/full_backup.sql | mysql -u <user> -p
or
shell> mysql -u <user> -p < /backups/full_backup.sql
```

To restore on the remote server, you can mention the `-h <hostname>` option:

```
shell> cat /backups/full_backup.sql | mysql -u <user> -p -h
<remote_hostname>
```

When you are restoring a backup, the backup statements will be logged to the binary log, which can slow down the restoration process. If you do not want the restoration process to write to the binary log, you can disable it at the session level using the `SET SQL_LOG_BIN=0;` option:

```
shell> (echo "SET SQL_LOG_BIN=0;";cat
/backups/full_backup.sql) | mysql -u <user> -p -h
<remote_hostname>
```

Or using:

```
mysql> SET SQL_LOG_BIN=0; SOURCE full_backup.sql
```

There's more...

1. Since backup restoration takes a very long time, it is recommended to start the restoration process inside a screen session so that even if you lose connectivity to the server, the restoration will continue.
2. Sometimes, there can be failures during restoration. If you pass the `--force` option to MySQL, restoration will continue:

```
shell> (echo "SET SQL_LOG_BIN=0;";cat
/backups/full_backup.sql) | mysql -u <user> -p -h
<remote_hostname> -f
```

Recovering from mydumper using myloader

`myloader` is a tool used for multi-threaded restoration of backups taken using `mydumper`. `myloader` comes along with `mydumper` and you don't need to install it separately. In this section, you will learn the various ways to restore a backup.

How to do it...

The common options for `myloader` are the hostname of the MySQL server to connect to (the default is `localhost`), username, password, and port.

Recovering full database

```
shell> myloader --directory=/backups --user=<user> --
password=<password> --queries-per-transaction=5000 --
threads=8 --compress-protocol --overwrite-tables
```

The options are explained as follows:

- `--overwrite-tables`: This option drops the tables if they already exist
- `--compress-protocol` : This option uses compression on the MySQL connection
- `--threads` : This option specifies the number of threads to use; the default is `4`
- `--queries-per-transaction` : This specifies the number of queries per transaction; the default is `1000`
- `--directory`: This specifies the directory of the dump to import

Recover a single database

You can specify `--source-db <db_name>` to restore only a single database.

Suppose you want to restore the `company` database:

```
shell> myloader --directory=/backups --queries-per-
transaction=5000 --threads=6 --compress-protocol --user=
<user> --password=<password> --source-db company --
overwrite-tables
```

Recovering a single table

`mydumper` writes the backup of each table to a separate `.sql` file. You can pick up the `.sql` file and restore:

```
shell> mysql -u <user> -p<password> -h <hostname> company -A
-f < company.payments.sql
```

If the table is split into chunks, you can copy all the chunks and information related to the table to a directory and specify the location.

Copy the required files:

```
shell> sudo cp
/backups/employee_table_chunks/employees.employees.* \
/backups/employee_table_chunks/employees.employees-
schema.sql \
/backups/employee_table_chunks/employees-schema-create.sql \
/backups/employee_table_chunks/metadata \
/backups/single_table/
```

Use `myloader` to load; it will automatically detect the chunks and load them:

```
shell> myloader --directory=/backups/single_table/ --
queries-per-transaction=50000 --threads=6 --compress-
protocol --overwrite-tables
```

Recovering from flat file backup

Recovering from flat files requires you to stop the MySQL server, replace all the files, change the permissions, and start MySQL.

How to do it...

1. Stop the MySQL server:

```
shell> sudo systemctl stop mysql
```

2. Move the files to the `data directory`:

```
shell> sudo mv /backup/mysql /var/lib
```

3. Change the ownership to `mysql`:

```
shell> sudo chown -R mysql:mysql /var/lib/mysql
```

4. Start MySQL:

```
shell> sudo systemctl start mysql
```

To minimize the downtime, if you have enough space on disk, you can copy to the backup to `/var/lib/mysql2`. Then stop MySQL, rename the directory, and start the server:

```
shell> sudo mv /backup/mysql /var/lib/mysql2
shell> sudo systemctl stop mysql
shell> sudo mv /var/lib/mysql2 /var/lib/mysql
shell> sudo chown -R mysql:mysql /var/lib/mysql
shell> sudo systemctl start mysql
```

Performing point-in-time recovery

Once the full backup is restored, you need to restore binary logs to get point-in-time recovery. The backups provide the binary log coordinates up to which the backups are available.

As explained in Chapter 7, *Backups*, in the *Locking instance for backup* section, you should choose the binary log backup from the right server, based on the `--dump-slave` or `--master-data` option specified in `mysqldump`.

How to do it...

Let's get into the details of doing it. There's a lot to learn here though.

mysqldump or mysqlpump

The binary log information is stored in the SQL file as the `CHANGE MASTER TO` command based on the options you passed to `mysqldump`/`mysqlpump`.

1. If you have used `--master-data`, you should use the binary logs of the slave:

```
shell> head -30 /backups/dump.sql
-- MySQL dump 10.13  Distrib 8.0.3-rc, for Linux
(x86_64)
--
-- Host: localhost    Database:
-- ------------------------------------------------------
--
-- Server version 8.0.3-rc-log
/*!40101 SET
@OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET
@OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET
@OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!50606 SET
@OLD_INNODB_STATS_AUTO_RECALC=@@INNODB_STATS_AUTO_RECAL
C */;
/*!50606 SET GLOBAL INNODB_STATS_AUTO_RECALC=OFF */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS,
UNIQUE_CHECKS=0 */;
/*!40014 SET
@OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0
*/;
```

```
--
-- Position to start replication or point-in-time
recovery from
--
CHANGE MASTER TO MASTER_LOG_FILE='server1.000008',
MASTER_LOG_POS=154;
```

In this case, you should start the restore from the `server1.000008` file at position `154` on the slave.

```
shell> mysqlbinlog --start-position=154 --disable-log-
bin /backups/binlogs/server1.000008 | mysql -u<user> -p
-h <host> -f
```

2. If you have used `--dump-slave`, you should use the binary logs of the master:

```
--
-- Position to start replication or point-in-time
recovery from (the master of this slave)
--
CHANGE MASTER TO MASTER_LOG_FILE='centos7-bin.000001',
MASTER_LOG_POS=463;
```

In this case, you should start the restore from the `centos7-bin.000001` file at position `463` located from the master.

```
shell> mysqlbinlog --start-position=463  --disable-log-bin
/backups/binlogs/centos7-bin.000001 | mysql -u<user> -p -h
<host> -f
```

mydumper

The binary log information is available in the metadata:

```
shell> sudo cat /backups/metadata
Started dump at: 2017-08-26 06:26:19
SHOW MASTER STATUS:
 Log: server1.000012
 Pos: 154
</span> GTID:
SHOW SLAVE STATUS:
 Host: 35.186.158.188
 Log: centos7-bin.000001
 Pos: 463
 GTID:
Finished dump at: 2017-08-26 06:26:24
```

If you have taken the binary log backup from the slave, you should start the restore from the `server1.000012` file at position `154` (`SHOW MASTER STATUS`):

```
shell> mysqlbinlog --start-position=154  --disable-log-bin
/backups/binlogs/server1.000012 | mysql -u<user> -p -h
<host> -f
```

If you have a binary log backup from the master, you should start the restore from `centos7-bin.000001` at position `463` (`SHOW SLAVE STATUS`):

```
shell> mysqlbinlog --start-position=463  --disable-log-bin
/backups/binlogs/centos7-bin.000001 | mysql -u<user> -p -h
<host> -f
```

Replication

In this chapter, we will cover the following recipes:

- Setting up replication
- Setting up master-master replication
- Setting up multi-source replication
- Setting up replication filters
- Switching a slave from master-slave to chain replication
- Switching a slave from chain replication to master-slave
- Setting up delayed replication
- Setting up GTID replication
- Setting up semi-synchronous replication

Introduction

As explained in Chapter 6, *Binary Logging*, replication enables data from one MySQL database server (the master) to be copied to one or more MySQL database servers (the slaves). Replication is asynchronous by default; slaves do not need to be permanently connected to receive updates from the master. You can configure to replicate all databases, selected databases, or even selected tables within a database.

In this chapter, you will learn how to set up traditional replication; replicate selected databases and tables; and set up multi-source replication, chain replication, delayed replication, and semi-synchronous replication.

On a high level, replication works like this: all DDL and DML statements executed on a server (**master**) are logged into binary logs, which are pulled by the servers connecting to it (called **slaves**). The binary logs are simply copied to the slaves and are saved as relay logs. This process is taken care of by a thread called **IO thread**. There is one more thread called **SQL thread**, that executes the statements in the relay log sequentially.

How replication works is very clearly explained in this blog:

https://www.percona.com/blog/2013/01/09/how-does-mysql-replication-really-work/

The advantages of replication are (taken from the manual, at https://dev.mysql.com/doc/refman/8.0/en/replication.html):

- **Scale-out solutions**: Spreading the load among multiple slaves to improve performance. In this environment, all writes and updates must take place on the master server. Reads, however, may take place on one or more slaves. This model can improve the performance of writes (since the master is dedicated to updates), while dramatically increasing read speed across an increasing number of slaves.

- **Data security**: Because data is replicated to the slave and the slave can pause the replication process, it is possible to run backup services on the slave without corrupting the corresponding master data.

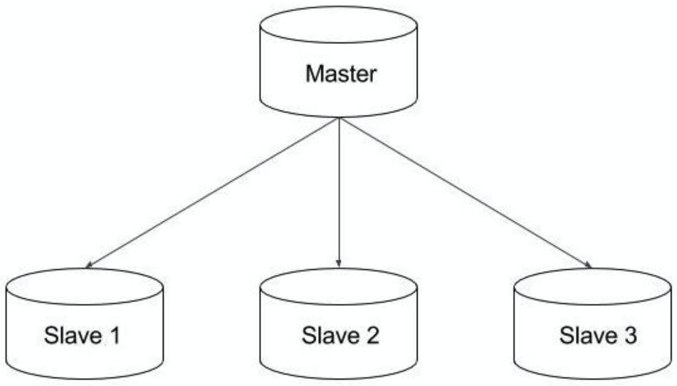
- **Analytics**: Live data can be created on the master, while the analysis of the information can take place on the slave without affecting the performance of the master.

- **Long-distance data distribution**: You can use replication to create a local copy of data for a remote site to use without permanent access to the master.

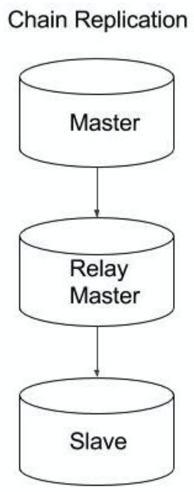
Setting up replication

There are many replication topologies. Some of them are the traditional master-slave replication, chain replication, master-master replication, multi-source replication, and so on.

Traditional replication involves a single master and multiple slaves.

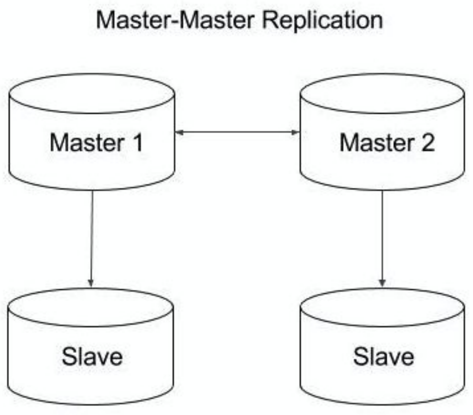


Chain replication means one server replicates from another, which in turn replicates from another. The middle server is referred to as the relay master (master ---> relay master ---> slave).

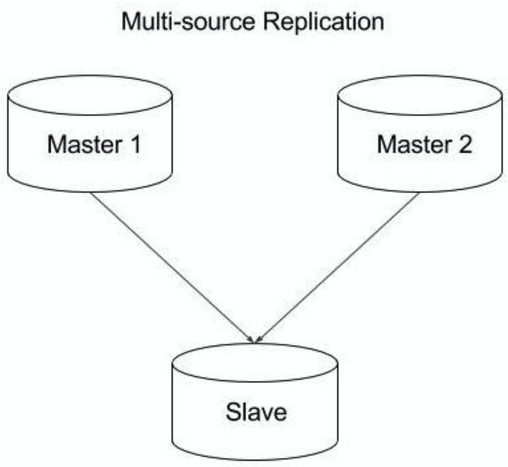


This is mainly used when you want to set up replication between two data centers. The primary master and its slaves will be in one data center. The secondary master (relay) replicates from the primary master in the other data center. All the slaves of the other data center are replicated from the secondary master.

Master-master replication: In this topology, both the masters accept writes and replicate between each other.



Multi-source replication: In this topology, a slave will replicate from multiple masters instead of one.



If you want to set up chain replication, you can follow the same steps mentioned here, replacing the master with the relay master.

How to do it...

In this section, setting up of single slave is explained. The same principles can be applied to set up chain replication. Usually the backups are taken from the slave when setting up another slave.

Outline:

1. Enable binary logging on the master
2. Create a replication user on the master
3. Set the unique `server_id` on the slave
4. Take backup from the master
5. Restore the backup on the slave
6. Execute the `CHANGE MASTER TO` command
7. Start the replication

Steps:

1. **On master**: Enable binary logging on the master and set `SERVER_ID`. Refer to Chapter 6, *Binary Logging*, to learn how to enable binary logging.
2. **On master**: Create a replication user. The slave connects to the master using this account:

```
mysql> GRANT REPLICATION SLAVE ON *.* TO
'binlog_user'@'%' IDENTIFIED BY 'binlog_P@ss12';
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

3. **On slave**: Set the unique `SERVER_ID` option (it should be different from what you have set on master):

```
mysql> SET @@GLOBAL.SERVER_ID = 32;
```

4. **On slave**: Take backup from the master by remotely connecting. You can use either `mysqldump` or `mydumper`. `mysqlpump` cannot be used because the binary log positions won't be consistent.

   `mysqldump`:

   ```
   shell> mysqldump -h <master_host> -u backup_user --
   password=<pass> --all-databases --routines --events --
   single-transaction --master-data  > dump.sql
   ```

   You have to pass the `--slave-dump` option when taking backup from another slave.

   `mydumper`:

   ```
   shell> mydumper -h <master_host> -u backup_user --
   password=<pass> --use-savepoints  --trx-consistency-
   only --kill-long-queries --outputdir /backups
   ```

5. **On slave**: After the backup completes, restore the backup. Refer to Chapter 8, *Restoring Data*, for the restoration methods.

   `mysqldump`:

   ```
   shell> mysql -u <user> -p -f < dump.sql
   ```

   `mydumper`:

   ```
   shell> myloader --directory=/backups --user=<user> --
   password=<password> --queries-per-transaction=5000 --
   threads=8 --overwrite-tables
   ```

6. **On slave**: After restoring the backup, you have to execute the following command:

   ```
   mysql> CHANGE MASTER TO MASTER_HOST='<master_host>',
   MASTER_USER='binlog_user',
   MASTER_PASSWORD='binlog_P@ss12',
   ```

```
MASTER_LOG_FILE='<log_file_name>', MASTER_LOG_POS=
<position>
```

`mysqldump`: `<log_file_name>` and `<position>` are included in the backup dump file. For example:

```
shell> less dump.sql
--
-- Position to start replication or point-in-time
recovery from (the master of this slave)
--
CHANGE MASTER TO MASTER_LOG_FILE='centos7-bin.000001',
MASTER_LOG_POS=463;
```

`mydumper`: `<log_file_name>` and `<position>` are stored in the metadata file:

```
shell> cat metadata
Started dump at: 2017-08-26 06:26:19
SHOW MASTER STATUS:
    Log: server1.000012
    Pos: 154122
    GTID:
SHOW SLAVE STATUS:
    Host: xx.xxx.xxx.xxx
    Log: centos7-bin.000001
    Pos: 463223
    GTID:
Finished dump at: 2017-08-26 06:26:24
```

If you are taking backup from a slave or master to set up another slave, you have to use positions from `SHOW SLAVE STATUS`. If you want to set up the chain replication, you can use the positions from `SHOW MASTER STATUS`.

1. On the slave, execute the `START SLAVE` command:

```
mysql> START SLAVE;
```

2. You can check the status of replication by executing:

```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row
***************************
               Slave_IO_State: Waiting for master to
send event
                  Master_Host: xx.xxx.xxx.xxx
                  Master_User: binlog_user
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: server1-bin.000001
          Read_Master_Log_Pos: 463
               Relay_Log_File: server2-relay-bin.000004
                Relay_Log_Pos: 322
        Relay_Master_Log_File: server1-bin.000001
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 463
              Relay_Log_Space: 1957
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: No
           Master_SSL_CA_File:
           Master_SSL_CA_Path:
              Master_SSL_Cert:
            Master_SSL_Cipher:
               Master_SSL_Key:
        Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 0
                Last_IO_Error:
               Last_SQL_Errno: 0
               Last_SQL_Error:
```

```
  Replicate_Ignore_Server_Ids:
             Master_Server_Id: 32
                  Master_UUID: b52ef45a-7ff4-11e7-9091-
42010a940003
             Master_Info_File:
/var/lib/mysql/master.info
                    SQL_Delay: 0
          SQL_Remaining_Delay: NULL
      Slave_SQL_Running_State: Slave has read all relay
log; waiting for more updates
           Master_Retry_Count: 86400
                  Master_Bind:
      Last_IO_Error_Timestamp:
     Last_SQL_Error_Timestamp:
               Master_SSL_Crl:
           Master_SSL_Crlpath:
           Retrieved_Gtid_Set:
            Executed_Gtid_Set:
                Auto_Position: 0
         Replicate_Rewrite_DB:
                 Channel_Name:
           Master_TLS_Version:
1 row in set (0.00 sec)
```

You should look for `Seconds_Behind_Master`, which shows the replication lag. If it is `0`, it means the slave is in sync with the master; any non-zero value indicates the number of seconds of lag, and if it is `NULL`, it means replication is not happening.

Setting up master-master replication

This recipe will interest many people since many of us try doing this. Let's get into the details of it.

How to do it...

Assume that the masters are `master1` and `master2`.

Steps:

1. Set up replication between `master1` and `master2` as described in Chapter 9, *Replication*.
2. Make `master2` read-only:

```
mysql> SET @@GLOBAL.READ_ONLY=ON;
```

3. On `master2`, check the current binary log coordinate.

```
mysql> SHOW MASTER STATUS;
+----------------+----------+--------------+-----------
-------+-------------------+
| File           | Position | Binlog_Do_DB |
Binlog_Ignore_DB | Executed_Gtid_Set |
+----------------+----------+--------------+-----------
-------+-------------------+
| server1.000017 |      473 |              |
|                  |
+----------------+----------+--------------+-----------
-------+-------------------+
1 row in set (0.00 sec)
```

   From the preceding output, you can start the replication on `master1` from `server1.000017` and position `473`.

4. From the positions taken from the preceding step, execute the `CHANGE MASTER TO` command on `master1`:

```
mysql> CHANGE MASTER TO MASTER_HOST='<master2_host>',
MASTER_USER='binlog_user',
MASTER_PASSWORD='binlog_P@ss12',
MASTER_LOG_FILE='<log_file_name>', MASTER_LOG_POS=
<position>
```

5. Start the slave on `master1`:

```
    mysql> START SLAVE;
```

6. Finally, you can make `master2` read-write, and applications can start writing to it.

```
     mysql> SET @@GLOBAL.READ_ONLY=OFF;
```

Setting up multi-source replication

MySQL multi-source replication enables a replication slave to receive transactions from multiple sources simultaneously. Multi-source replication can be used to back up multiple servers to a single server, merge table shards, and consolidate data from multiple servers to a single server. Multi-source replication does not implement any conflict detection or resolution when applying transactions, and those tasks are left to the application if required. In a multi-source replication topology, a slave creates a replication channel for each master that it should receive transactions from.

In this section, you will learn how to set up a slave with multiple masters. This method is the same as setting up traditional replication over the channels.

How to do it...

Assume that you are setting up `server3` as a slave of `server1` and `server2`. You need to create traditional replication from `server1` to `server3` over a channel and from `server2` to `server3` over another channel. To ensure that data is consistent on the slave, make sure that different sets of databases are replicated or the application takes care of the conflicts.

Before you begin, take a backup from server1 and restore on `server3`; similarly take a backup from `server2` and restore on `server3`, as described in Chapter 9, *Replication*.

1. On `server3`, modify the replication repositories to `TABLE` from `FILE`. You can change it dynamically by running the following commands:

```
mysql> STOP SLAVE; //If slave is already running
mysql> SET GLOBAL master_info_repository = 'TABLE';
mysql> SET GLOBAL relay_log_info_repository = 'TABLE';
```

   Also make the changes in the configuration file:

```
shell> sudo vi /etc/my.cnf
[mysqld]
master-info-repository=TABLE
relay-log-info-repository=TABLE
```

2. On `server3`, execute the `CHANGE MASTER TO` command to make it a slave of `server1` over a channel named `master-1`. You can name it anything:

```
mysql> CHANGE MASTER TO MASTER_HOST='server1',
MASTER_USER='binlog_user', MASTER_PORT=3306,
MASTER_PASSWORD='binlog_P@ss12',
MASTER_LOG_FILE='server1.000017', MASTER_LOG_POS=788
FOR CHANNEL 'master-1';
```

3. On `server3`, execute the `CHANGE MASTER TO` command to make it a slave of `server2` over channel `master-2`:

```
mysql> CHANGE MASTER TO MASTER_HOST='server2',
MASTER_USER='binlog_user', MASTER_PORT=3306,
MASTER_PASSWORD='binlog_P@ss12',
MASTER_LOG_FILE='server2.000014', MASTER_LOG_POS=75438
FOR CHANNEL 'master-2';
```

4. Execute the `START SLAVE FOR CHANNEL` statement for each channel as follows:

```
mysql> START SLAVE FOR CHANNEL 'master-1';
Query OK, 0 rows affected (0.01 sec)

mysql> START SLAVE FOR CHANNEL 'master-2';
Query OK, 0 rows affected (0.00 sec)
```

5. Verify the slave status by executing the `SHOW SLAVE STATUS` statement:

```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row
***************************
               Slave_IO_State: Waiting for master to
send event
                  Master_Host: server1
                  Master_User: binlog_user
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: server1.000017
          Read_Master_Log_Pos: 788
               Relay_Log_File: server3-relay-bin-
master@002d1.000002
                Relay_Log_Pos: 318
        Relay_Master_Log_File: server1.000017
             Slave_IO_Running: Yes
```

```
              Slave_SQL_Running: Yes
                Replicate_Do_DB:
            Replicate_Ignore_DB:
             Replicate_Do_Table:
         Replicate_Ignore_Table:
        Replicate_Wild_Do_Table:
    Replicate_Wild_Ignore_Table:
                     Last_Errno: 0
                     Last_Error:
                   Skip_Counter: 0
            Exec_Master_Log_Pos: 788
                Relay_Log_Space: 540
                Until_Condition: None
                 Until_Log_File:
                  Until_Log_Pos: 0
             Master_SSL_Allowed: No
             Master_SSL_CA_File:
             Master_SSL_CA_Path:
                Master_SSL_Cert:
              Master_SSL_Cipher:
                 Master_SSL_Key:
          Seconds_Behind_Master: 0
  Master_SSL_Verify_Server_Cert: No
                  Last_IO_Errno: 0
                  Last_IO_Error:
                 Last_SQL_Errno: 0
                 Last_SQL_Error:
    Replicate_Ignore_Server_Ids:
               Master_Server_Id: 32
                    Master_UUID: 7cc7fca7-4deb-11e7-a53e-
42010a940002
               Master_Info_File: mysql.slave_master_info
                      SQL_Delay: 0
            SQL_Remaining_Delay: NULL
        Slave_SQL_Running_State: Slave has read all relay
log; waiting for more updates
             Master_Retry_Count: 86400
                    Master_Bind:
        Last_IO_Error_Timestamp:
       Last_SQL_Error_Timestamp:
                 Master_SSL_Crl:
             Master_SSL_Crlpath:
             Retrieved_Gtid_Set:
              Executed_Gtid_Set:
                  Auto_Position: 0
           Replicate_Rewrite_DB:
                   Channel_Name: master-1
             Master_TLS_Version:
  *************************** 2. row
  ***************************
                 Slave_IO_State: Waiting for master to
  send event
                    Master_Host: server2
                    Master_User: binlog_user
                    Master_Port: 3306
                  Connect_Retry: 60
                Master_Log_File: server2.000014
            Read_Master_Log_Pos: 75438
                 Relay_Log_File: server3-relay-bin-
  master@002d2.000002
                  Relay_Log_Pos: 322
          Relay_Master_Log_File: server2.000014
               Slave_IO_Running: Yes
              Slave_SQL_Running: Yes
                Replicate_Do_DB:
            Replicate_Ignore_DB:
             Replicate_Do_Table:
         Replicate_Ignore_Table:
        Replicate_Wild_Do_Table:
    Replicate_Wild_Ignore_Table:
                     Last_Errno: 0
                     Last_Error:
                   Skip_Counter: 0
            Exec_Master_Log_Pos: 75438
                Relay_Log_Space: 544
                Until_Condition: None
                 Until_Log_File:
                  Until_Log_Pos: 0
             Master_SSL_Allowed: No
             Master_SSL_CA_File:
             Master_SSL_CA_Path:
                Master_SSL_Cert:
              Master_SSL_Cipher:
                 Master_SSL_Key:
          Seconds_Behind_Master: 0
  Master_SSL_Verify_Server_Cert: No
                  Last_IO_Errno: 0
                  Last_IO_Error:
                 Last_SQL_Errno: 0
                 Last_SQL_Error:
```

```
  Replicate_Ignore_Server_Ids:
             Master_Server_Id: 32
                  Master_UUID: b52ef45a-7ff4-11e7-9091-
42010a940003
             Master_Info_File: mysql.slave_master_info
                    SQL_Delay: 0
          SQL_Remaining_Delay: NULL
      Slave_SQL_Running_State: Slave has read all relay
log; waiting for more updates
           Master_Retry_Count: 86400
                  Master_Bind:
      Last_IO_Error_Timestamp:
     Last_SQL_Error_Timestamp:
               Master_SSL_Crl:
           Master_SSL_Crlpath:
           Retrieved_Gtid_Set:
            Executed_Gtid_Set:
                Auto_Position: 0
         Replicate_Rewrite_DB:
                 Channel_Name: master-2
           Master_TLS_Version:
2 rows in set (0.00 sec)
```

6. To get the slave status for a particular channel, execute:

```
mysql> SHOW SLAVE STATUS FOR CHANNEL 'master-1' \G
```

7. This is the other way you can use a performance schema to monitor the metrics:

```
mysql> SELECT * FROM
performance_schema.replication_connection_status\G
*************************** 1. row
***************************
                                      CHANNEL_NAME:
master-1
                                        GROUP_NAME:
                                       SOURCE_UUID:
7cc7fca7-4deb-11e7-a53e-42010a940002
                                         THREAD_ID: 36
                                     SERVICE_STATE: ON
                         COUNT_RECEIVED_HEARTBEATS: 73
                          LAST_HEARTBEAT_TIMESTAMP:
```

```
2017-09-15 12:42:10.910051
                         RECEIVED_TRANSACTION_SET:
                                LAST_ERROR_NUMBER: 0
                               LAST_ERROR_MESSAGE:
                             LAST_ERROR_TIMESTAMP:
0000-00-00 00:00:00.000000
                          LAST_QUEUED_TRANSACTION:
 LAST_QUEUED_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP:
0000-00-00 00:00:00.000000
LAST_QUEUED_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP:
0000-00-00 00:00:00.000000
     LAST_QUEUED_TRANSACTION_START_QUEUE_TIMESTAMP:
0000-00-00 00:00:00.000000
       LAST_QUEUED_TRANSACTION_END_QUEUE_TIMESTAMP:
0000-00-00 00:00:00.000000
                             QUEUEING_TRANSACTION:
    QUEUEING_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP:
0000-00-00 00:00:00.000000
   QUEUEING_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP:
0000-00-00 00:00:00.000000
        QUEUEING_TRANSACTION_START_QUEUE_TIMESTAMP:
0000-00-00 00:00:00.000000
*************************** 2. row
***************************
                                     CHANNEL_NAME:
master-2
                                       GROUP_NAME:
                                      SOURCE_UUID:
b52ef45a-7ff4-11e7-9091-42010a940003
                                        THREAD_ID: 38
                                    SERVICE_STATE: ON
                        COUNT_RECEIVED_HEARTBEATS: 73
                         LAST_HEARTBEAT_TIMESTAMP:
2017-09-15 12:42:13.986271
                         RECEIVED_TRANSACTION_SET:
                                LAST_ERROR_NUMBER: 0
                               LAST_ERROR_MESSAGE:
                             LAST_ERROR_TIMESTAMP:
0000-00-00 00:00:00.000000
                          LAST_QUEUED_TRANSACTION:
 LAST_QUEUED_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP:
0000-00-00 00:00:00.000000
LAST_QUEUED_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP:
0000-00-00 00:00:00.000000
     LAST_QUEUED_TRANSACTION_START_QUEUE_TIMESTAMP:
```

```
0000-00-00 00:00:00.000000
       LAST_QUEUED_TRANSACTION_END_QUEUE_TIMESTAMP:
0000-00-00 00:00:00.000000
                              QUEUEING_TRANSACTION:
    QUEUEING_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP:
0000-00-00 00:00:00.000000
   QUEUEING_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP:
0000-00-00 00:00:00.000000
        QUEUEING_TRANSACTION_START_QUEUE_TIMESTAMP:
0000-00-00 00:00:00.000000
2 rows in set (0.00 sec)
```

You can specify all the slave-related commands for a channel by appending `FOR CHANNEL 'channel_name'`:

```
mysql> STOP SLAVE FOR CHANNEL 'master-1';
mysql> RESET SLAVE FOR CHANNEL 'master-2';
```

Setting up replication filters

You can control which tables or databases are to be replicated. On the master, you can control which databases to log changes for by using the `--binlog-do-db` and `--binlog-ignore-db` options to control binary logging, as mentioned in Chapter 6, *Binary Logging*. The better way is to control on the slave side. You can execute or ignore statements received from the master by using `--replicate-*` options or dynamically by creating replication filters.

How to do it...

To create a filter, you need to execute the `CHANGE REPLICATION FILTER` statement.

Replicate a database only

Assume that you want to replicate `db1` and `db2` only. Use the following statement to create the replication filter.

```
mysql> CHANGE REPLICATION FILTER REPLICATE_DO_DB = (db1,
db2);
```

Note that you should specify all the databases inside parentheses.

Replicate specific tables

You can specify the tables you want to be replicated using `REPLICATE_DO_TABLE`:

```
mysql> CHANGE REPLICATION FILTER REPLICATE_DO_TABLE =
('db1.table1');
```

Suppose that you want to use regex for tables; you can use the `REPLICATE_WILD_DO_TABLE` option:

```
mysql> CHANGE REPLICATION FILTER REPLICATE_WILD_DO_TABLE =
('db1.imp%');
```

You can mention some databases or tables with regex using various `IGNORE` options.

Ignore a database

Just like you can choose to replicate a database, you can ignore a database from replication using `REPLICATE_IGNORE_DB`:

```
mysql> CHANGE REPLICATION FILTER REPLICATE_IGNORE_DB = (db1,
db2);
```

Ignore specific tables

You can ignore certain tables using the `REPLICATE_IGNORE_TABLE` and `REPLICATE_WILD_IGNORE_TABLE` options.
The `REPLICATE_WILD_IGNORE_TABLE` option allows wildcard characters, where as `REPLICATE_IGNORE_TABLE` only accepts full table names:

```
mysql> CHANGE REPLICATION FILTER REPLICATE_IGNORE_TABLE =
('db1.table1');
mysql> CHANGE REPLICATION FILTER REPLICATE_WILD_IGNORE_TABLE
= ('db1.new%', 'db2.new%');
```

You can also set filters for a channel by specifying the channel name:

```
mysql> CHANGE REPLICATION FILTER REPLICATE_DO_DB = (d1) FOR
CHANNEL 'master-1';
```

See also

Refer to https://dev.mysql.com/doc/refman/8.0/en/change-replication-filter.html for more details on replication filters. If you are using more than one filter, refer to https://dev.mysql.com/doc/refman/8.0/en/replication-rules.html to know more about how MySQL evaluates filters.

Switching slave from master-slave to chain replication

If you have a master-slave replication set up, Servers B and C replicating from Server A: Server A --> (Server B, Server C) and you wish to make Server C a slave of Server B, you have to stop replication on both Server B and Server C. Then bring them to the same master log position using the `START SLAVE UNTIL` command. After that, you can get the master log coordinates from Server B and execute the `CHANGE MASTER TO` command on Server C.

How to do it...

1. **On Server C**: Stop slave and note the `Relay_Master_Log_File` and `Exec_Master_Log_Pos` positions in the `SHOW SLAVE STATUS\G` command:

```
mysql> STOP SLAVE;
Query OK, 0 rows affected (0.01 sec)

mysql> SHOW SLAVE STATUS\G
*************************** 1. row
***************************
               Slave_IO_State:
                  Master_Host: xx.xxx.xxx.xxx
                  Master_User: binlog_user
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: server_A-bin.000023
          Read_Master_Log_Pos: 2604
               Relay_Log_File: server_C-relay-
bin.000002
                Relay_Log_Pos: 1228
        Relay_Master_Log_File: server_A-bin.000023
~
          Exec_Master_Log_Pos: 2604
              Relay_Log_Space: 1437
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
~
1 row in set (0.00 sec)
```

2. **On Server B**: Stop slave and note the `Relay_Master_Log_File` and `Exec_Master_Log_Pos` positions in the `SHOW SLAVE STATUS\G` command:

```
mysql> STOP SLAVE;
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row
***************************
               Slave_IO_State:
                  Master_Host: xx.xxx.xxx.xxx
                  Master_User: binlog_user
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: server_A-bin.000023
          Read_Master_Log_Pos: 8250241
               Relay_Log_File: server_B-relay-
bin.000002
                Relay_Log_Pos: 1228
        Relay_Master_Log_File: server_A-bin.000023
~
          Exec_Master_Log_Pos: 8250241
              Relay_Log_Space: 8248167
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
~
1 row in set (0.00 sec)
```

3. Compare the Server B log positions with Server C and find out which is the latest sync with Server A. Usually, since you have stopped the slave on Server C first, Server B will be ahead. In our case, the log positions are:
   Server C: (`server_A-bin.000023`, `2604`)
   Server B: (`server_A-bin.000023`, `8250241`)
   Server B is ahead, so we have to bring Server C to the position of Server B.
4. **On Server C**: Use the `START SLAVE UNTIL` statement to sync up to the position of server B:

```
mysql> START SLAVE UNTIL MASTER_LOG_FILE='centos7-
bin.000023', MASTER_LOG_POS=8250241;
Query OK, 0 rows affected, 1 warning (0.03 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row
```

```
***************************
  Level: Note
   Code: 1278
Message: It is recommended to use --skip-slave-start
when doing step-by-step replication with START SLAVE
UNTIL; otherwise, you will get problems if you get an
unexpected slave's mysqld restart
1 row in set (0.00 sec)
```

5. **On Server C**: Wait for server C to catch up by checking `Exec_Master_Log_Pos` and `Until_Log_Pos` (both should be the same) in the output of `SHOW SLAVE STATUS`:

```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row
***************************
               Slave_IO_State: Waiting for master to
send event
                  Master_Host: xx.xxx.xxx.xxx
                  Master_User: binlog_user
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: server_A-bin.000023
          Read_Master_Log_Pos: 8250241
               Relay_Log_File: server_C-relay-
bin.000003
                Relay_Log_Pos: 8247959
        Relay_Master_Log_File: server_A-bin.000023
             Slave_IO_Running: Yes
            Slave_SQL_Running: No
~
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 8250241
              Relay_Log_Space: 8249242
              Until_Condition: Master
               Until_Log_File: server_A-bin.000023
                Until_Log_Pos: 8250241
           Master_SSL_Allowed: No
           Master_SSL_CA_File:
           Master_SSL_CA_Path:
              Master_SSL_Cert:
```

```
            Master_SSL_Cipher:
               Master_SSL_Key:
        Seconds_Behind_Master: NULL
~
1 row in set (0.00 sec)
```

6. **On Server B**: Find out the master status, start the slave, and make sure it is replicating:

```
mysql> SHOW MASTER STATUS;
+----------------------+----------+--------------+------
------------+-------------------+
| File                 | Position | Binlog_Do_DB |
Binlog_Ignore_DB | Executed_Gtid_Set |
+----------------------+----------+--------------+------
------------+-------------------+
| server_B-bin.000003 | 36379324 |              |
|                   |
+----------------------+----------+--------------+------
------------+-------------------+
1 row in set (0.00 sec)

mysql> START SLAVE;
Query OK, 0 rows affected (0.02 sec)

mysql> SHOW SLAVE STATUS\G
*************************** 1. row
***************************
               Slave_IO_State:
                  Master_Host: xx.xxx.xxx.xxx
                  Master_User: binlog_user
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: server_A-bin.000023
          Read_Master_Log_Pos: 8250241
               Relay_Log_File: server_B-relay-
bin.000002
                Relay_Log_Pos: 1228
        Relay_Master_Log_File: server_A-bin.000023
~
          Exec_Master_Log_Pos: 8250241
              Relay_Log_Space: 8248167
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
~
1 row in set (0.00 sec)
```

7. **On Server C**: Stop the slave, execute the `CHANGE MASTER TO` command, and point to server B. You have to use the positions that you have got from the preceding step:

```
mysql> STOP SLAVE;
Query OK, 0 rows affected (0.04 sec)

mysql> CHANGE MASTER TO MASTER_HOST = 'Server B',
MASTER_USER = 'binlog_user', MASTER_PASSWORD =
'binlog_P@ss12', MASTER_LOG_FILE='server_B-bin.000003',
MASTER_LOG_POS=36379324;
Query OK, 0 rows affected, 1 warning (0.04 sec)
```

8. **On Server C**: Start replication and verify the slave status:

```
mysql> START SLAVE;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW SLAVE STATUS\G
Query OK, 0 rows affected, 1 warning (0.00 sec)

*************************** 1. row
***************************
               Slave_IO_State: Waiting for master to
send event
                  Master_Host: xx.xxx.xxx.xx
                  Master_User: binlog_user
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: server_B-bin.000003
          Read_Master_Log_Pos: 36380416
               Relay_Log_File: server_C-relay-
bin.000002
                Relay_Log_Pos: 1413
        Relay_Master_Log_File: server_B-bin.000003
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
 ~
          Exec_Master_Log_Pos: 36380416
```

```
              Relay_Log_Space: 1622
 ~
        Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 0
                Last_IO_Error:
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
~
1 row in set (0.00 sec)
```

Switching the slave from chain replication to master-slave

If you have a chain replication setup (such as server A --> sever B --> server C) and you wish to make server C a direct slave of server A, you have to stop replication on server B, let server C catch up with server B, and then find the coordinates of server A corresponding to the position where server B stopped. Using those coordinates, you can execute a `CHANGE MASTER TO` command on server C and make it a slave of server A.

How to do it...

1. **On server B**: Stop the slave and note down the master status:

```
mysql> STOP SLAVE;
Query OK, 0 rows affected (0.04 sec)

mysql> SHOW MASTER STATUS;
+----------------------+----------+--------------+------
------------+-------------------+
| File                 | Position | Binlog_Do_DB |
Binlog_Ignore_DB | Executed_Gtid_Set |
+----------------------+----------+--------------+------
------------+-------------------+
| server_B-bin.000003 | 44627878 |              |
|                   |
+----------------------+----------+--------------+------
------------+-------------------+
1 row in set (0.00 sec)
```

2. **On server C**: Make sure that the slave delay is caught up. `Relay_Master_Log_File` and `Exec_Master_Log_Pos` should be equal to the output of the master status on server B. Once the delay is caught up, stop the slave:

```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row
***************************
               Slave_IO_State: Waiting for master to
send event
                  Master_Host: 35.186.157.16
                  Master_User: repl
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: server_B-bin.000003
          Read_Master_Log_Pos: 44627878
               Relay_Log_File: ubuntu2-relay-bin.000002
                Relay_Log_Pos: 8248875
        Relay_Master_Log_File: server_B-bin.000003
```

```
                   Slave_IO_Running: Yes
                  Slave_SQL_Running: Yes
                    Replicate_Do_DB:
                Replicate_Ignore_DB:
                 Replicate_Do_Table:
             Replicate_Ignore_Table:
            Replicate_Wild_Do_Table:
        Replicate_Wild_Ignore_Table:
                         Last_Errno: 0
                         Last_Error:
                       Skip_Counter: 0
                **Exec_Master_Log_Pos: 44627878**
                    Relay_Log_Space: 8249084
                    Until_Condition: None
                     Until_Log_File:
                      Until_Log_Pos: 0
                 Master_SSL_Allowed: No
                 Master_SSL_CA_File:
                 Master_SSL_CA_Path:
                    Master_SSL_Cert:
                  Master_SSL_Cipher:
                     Master_SSL_Key:
              **Seconds_Behind_Master: 0**
~
1 row in set (0.00 sec)
```

```
mysql> STOP SLAVE;
Query OK, 0 rows affected (0.01 sec)
```

3. **On server B**: Get the coordinates of server A from the SHOW SLAVE STATUS output (note down Relay_Master_Log_File and Exec_Master_Log_Pos) and start the slave:

```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row
***************************
                     Slave_IO_State:
                        Master_Host: xx.xxx.xxx.xxx
                        Master_User: repl
                        Master_Port: 3306
                      Connect_Retry: 60
                    Master_Log_File: server_A-bin.000023
                Read_Master_Log_Pos: 16497695
                     Relay_Log_File: server_B-relay-
```

```
bin.000004
                Relay_Log_Pos: 8247776
        Relay_Master_Log_File: server_A-bin.000023
             Slave_IO_Running: No
            Slave_SQL_Running: No
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 16497695
              Relay_Log_Space: 8248152
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: No
           Master_SSL_CA_File:
           Master_SSL_CA_Path:
              Master_SSL_Cert:
            Master_SSL_Cipher:
               Master_SSL_Key:
        Seconds_Behind_Master: NULL
```

```
mysql> START SLAVE;
Query OK, 0 rows affected (0.01 sec)
```

4. **On server C**: Stop the slave and execute `CHANGE MASTER TO COMMAND` to point to server A. Use the positions noted down from the preceding step (`server_A-bin.000023` and `16497695`). Finally start the slave and verify the slave status:

```
mysql> STOP SLAVE;
Query OK, 0 rows affected (0.07 sec)
```

```
mysql> CHANGE MASTER TO MASTER_HOST = 'Server A',
MASTER_USER = 'binlog_user', MASTER_PASSWORD =
'binlog_P@ss12', MASTER_LOG_FILE='server_A-bin.000023',
MASTER_LOG_POS=16497695;
Query OK, 0 rows affected, 1 warning (0.02 sec)
```

```
mysql> START SLAVE;
Query OK, 0 rows affected (0.07 sec)

mysql> SHOW SLAVE STATUS\G
*************************** 1. row
***************************
               Slave_IO_State:
                  Master_Host: xx.xxx.xxx.xxx
                  Master_User: binlog_user
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: server_A-bin.000023
          Read_Master_Log_Pos: 16497695
               Relay_Log_File: server_C-relay-
bin.000001
                Relay_Log_Pos: 4
        Relay_Master_Log_File: server_A-bin.000023
             Slave_IO_Running: No
            Slave_SQL_Running: No
  ~
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 16497695
              Relay_Log_Space: 154
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: No
           Master_SSL_CA_File:
           Master_SSL_CA_Path:
              Master_SSL_Cert:
            Master_SSL_Cipher:
               Master_SSL_Key:
        Seconds_Behind_Master: 0
~
1 row in set (0.00 sec)
```

Setting up delayed replication

Sometimes, you need a delayed slave for disaster recovery purpose. Suppose a disastrous statement (such as a `DROP DATABASE` command) was executed on the master. You have to use *point-in-time recovery* from backups to restore the database. It will lead to a huge downtime depending on the size of the database. To avoid that situation, you can use a delayed slave, which will be always delayed from the master by a configured amount of time. If a disaster occurs and that statement is not applied by the delayed slave, you can stop the slave and start until the disastrous statement, so that the disastrous statement won't be executed. Then promote it to master.

The procedure is exactly the same as setting up normal replication, except that you specify `MASTER_DELAY` in the `CHANGE MASTER TO` command.

How is the delay measured?

In versions earlier than MySQL 8.0, the delay is measured based on the `Seconds_Behind_Master` value. In MySQL 8.0, it is measured based on `original_commit_timestamp` and `immediate_commit_timestamp`, which are written to the binary log.

`original_commit_timestamp` is the number of microseconds since the epoch when the transaction was written (committed) to the binary log of the original master.

`immediate_commit_timestamp` is the number of microseconds since the epoch when the transaction was written (committed) to the binary log of the immediate master.

How to do it...

1. Stop the slave:

```
mysql> STOP SLAVE;
Query OK, 0 rows affected (0.06 sec)
```

2. Execute `CHANGE MASTER TO MASTER_DELAY =` and start the slave. Suppose you want a 1-hour delay, you can set `MASTER_DELAY` to `3600` seconds:

```
mysql> CHANGE MASTER TO MASTER_DELAY = 3600;
Query OK, 0 rows affected (0.04 sec)

mysql> START SLAVE;
Query OK, 0 rows affected (0.00 sec)
```

3. Check for the following in the slave status:
   `SQL_Delay`: Number of seconds by which the slave must lag the master.
   `SQL_Remaining_Delay`: Number of seconds left of the delay. This is NULL when there is delay is maintained.
   `Slave_SQL_Running_State`: The state of the SQL thread.

```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row
***************************
               Slave_IO_State: Waiting for master to
send event
                  Master_Host: 35.186.158.188
                  Master_User: repl
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: server_A-bin.000023
          Read_Master_Log_Pos: 24745149
               Relay_Log_File: server_B-relay-
bin.000002
```

```
                  Relay_Log_Pos: 322
          Relay_Master_Log_File: server_A-bin.000023
               Slave_IO_Running: Yes
              Slave_SQL_Running: Yes
~
                     Last_Errno: 0
                     Last_Error:
                   Skip_Counter: 0
            Exec_Master_Log_Pos: 16497695
                Relay_Log_Space: 8247985
                Until_Condition: None
                 Until_Log_File:
                  Until_Log_Pos: 0
~
          Seconds_Behind_Master: 52
Master_SSL_Verify_Server_Cert: No
                  Last_IO_Errno: 0
                  Last_IO_Error:
                 Last_SQL_Errno: 0
                 Last_SQL_Error:
~
                      SQL_Delay: 3600
            SQL_Remaining_Delay: 3549
        Slave_SQL_Running_State: Waiting until
MASTER_DELAY seconds after master executed event
             Master_Retry_Count: 86400
                    Master_Bind:
        Last_IO_Error_Timestamp:
       Last_SQL_Error_Timestamp:
~
1 row in set (0.00 sec)
```

Note that `Seconds_Behind_Master` will be shown as `0` once the delay is maintained.

Setting up GTID replication

A **global transaction identifier** (**GTID**) is a unique identifier created and associated with each transaction committed on the server of origin (master). This identifier is unique, not only to the server on which it originated, but also across all servers in a given replication setup. There is a one-to-one mapping between all transactions and all GTIDs.

A GTID is represented as a pair of coordinates, separated by a colon character (`:`):

```
GTID = source_id:transaction_id
```

The `source_id` option identifies the originating server. Normally, the server's `server_uuid` option is used for this purpose. The `transaction_id` option is a sequence number determined by the order in which the transaction was committed on this server. For example, the first transaction to be committed has `1` as its `transaction_id`, and the tenth transaction to be committed on the same originating server is assigned a `transaction_id` of `10`.

As you have seen in previous methods, you have to mention the binary log file and position as the starting point for replication. If you are switching a slave from one master to another, especially during a failover, you have to get the positions from the new master to sync the slave, which can be painful. To avoid these, you can use GTID-based replication, where MySQL automatically detects binary log positions using GTIDs.

How to do it...

If the replication is already set up between the servers, follow these steps:

1. Enable GTIDs in `my.cnf`:

   ```
   shell> sudo vi /etc/my.cnf
   [mysqld]
   gtid_mode=ON
   enforce-gtid-consistency=true
   skip_slave_start
   ```

2. Set the master as read-only and make sure that all the slaves catch up with the master. This is very important because there should not be any data inconsistency between master and slaves:

   ```
   On master
   mysql> SET @@global.read_only = ON;

   On Slaves (if replication is already setup)
   mysql> SHOW SLAVE STATUS\G
   ```

3. Restart all the slaves to put GTID into effect. Since the `skip_slave_start` is given in the configuration file, the slave won't start until you specify the `START SLAVE` command. If you start the slave, it will fail with this error—`The replication receiver thread cannot start because the master has GTID_MODE = OFF and this server has GTID_MODE = ON`:

   ```
   shell> sudo systemctl restart mysql
   ```

4. Restart the master. When you restart the master, it begins in read-write mode and starts accepting writes in GTID mode:

```
shell> sudo systemctl restart mysql
```

5. Execute the `CHANGE MASTER TO` command to set up GTID replication:

```
mysql> CHANGE MASTER TO MASTER_HOST = <master_host>,
MASTER_PORT = <port>, MASTER_USER = 'binlog_user',
MASTER_PASSWORD = 'binlog_P@ss12', MASTER_AUTO_POSITION
= 1;
```

   You can observe that the binary log file and positions are not given; instead, `MASTER_AUTO_POSITION` is given, which automatically finds the GTIDs executed.

6. Execute `START SLAVE` on all slaves:

```
mysql> START SLAVE;
```

7. Verify that the slave is replicating:

```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row
***************************
               Slave_IO_State: Waiting for master to
send event
                  Master_Host: xx.xxx.xxx.xxx
                  Master_User: binlog_user
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: server1-bin.000002
          Read_Master_Log_Pos: 345
               Relay_Log_File: server2-relay-bin.000002
                Relay_Log_Pos: 562
        Relay_Master_Log_File: server1-bin.000002
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Replicate_Do_DB:
```

```
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 345
              Relay_Log_Space: 770
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: No
           Master_SSL_CA_File:
           Master_SSL_CA_Path:
              Master_SSL_Cert:
            Master_SSL_Cipher:
               Master_SSL_Key:
        Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 0
                Last_IO_Error:
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Master_Server_Id: 32
                  Master_UUID: b52ef45a-7ff4-11e7-9091-
42010a940003
             Master_Info_File:
/var/lib/mysql/master.info
                    SQL_Delay: 0
          SQL_Remaining_Delay: NULL
      Slave_SQL_Running_State: Slave has read all relay
log; waiting for more updates
           Master_Retry_Count: 86400
                  Master_Bind:
      Last_IO_Error_Timestamp:
     Last_SQL_Error_Timestamp:
               Master_SSL_Crl:
           Master_SSL_Crlpath:
           Retrieved_Gtid_Set: b52ef45a-7ff4-11e7-9091-
42010a940003:1
            Executed_Gtid_Set: b52ef45a-7ff4-11e7-9091-
42010a940003:1
```

```
                Auto_Position: 1
         Replicate_Rewrite_DB:
                 Channel_Name:
           Master_TLS_Version:
1 row in set (0.00 sec)
```

To know more about GTID, refer to https://dev.mysql.com/doc/refman/5.6/en/replication-gtids-concepts.html.

Setting up semi-synchronous replication

Replication is asynchronous by default. The master is not aware of whether the writes have reached the slaves or not. If there is a delay between master and slave, and if the master crashes, you will lose the data that has not reached the slave. To overcome this situation, you can use semi-synchronous replication.

In semi-synchronous replication, the master waits until at least one slave has received the writes. By default, the value of `rpl_semi_sync_master_wait_point` is `AFTER_SYNC`; this means that the master syncs the transaction to the binary log, which is consumed by the slave.

After that, the slave sends an acknowledgement to the master, then the master commits the transaction and returns the result to the client. So, it is enough if the writes have reached the relay log; the slave need not commit the transaction. You can change this behavior by changing the variable `rpl_semi_sync_master_wait_point` to `AFTER_COMMIT`. In this, the master commits the transaction to the storage engine but does not return the result to the client. Once the transaction is committed on the slave, the master receives an acknowledgment of transaction and then returns a result to the client.

If you want the transaction to be acknowledged on more slaves, you can increase the value of the dynamic variable `rpl_semi_sync_master_wait_for_slave_count`. You can also set how many milliseconds the master has to wait for to get the acknowledgement from the slave through the dynamic variable `rpl_semi_sync_master_timeout`; the default is `10` seconds.

In fully synchronous replication, the master waits until all the slaves have committed the transaction. To implement this, you have to use Galera Cluster.

How to do it...

On a high level, you need to install and enable semi-synchronous plugins on both the master and all slaves where you want semi-synchronous replication. You have to restart the slave IO thread to bring the changes into effect. You can adjust the value of `rpl_semi_sync_master_timeout` according to your network and application. A value of `1` second is a good start:

1. On the master, install the `rpl_semi_sync_master` plugin:

   ```
   mysql> INSTALL PLUGIN rpl_semi_sync_master SONAME
   'semisync_master.so';
   Query OK, 0 rows affected (0.86 sec)
   ```

   Verify that the plugin is activated:

   ```
   mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS FROM
   INFORMATION_SCHEMA.PLUGINS WHERE PLUGIN_NAME LIKE
   '%semi%';
   +----------------------+---------------+
   | PLUGIN_NAME          | PLUGIN_STATUS |
   +----------------------+---------------+
   | rpl_semi_sync_master | ACTIVE        |
   +----------------------+---------------+
   1 row in set (0.01 sec)
   ```

2. On the master, enable semi-synchronous replication and adjust the timeout (say 1 second):

   ```
   mysql> SET @@GLOBAL.rpl_semi_sync_master_enabled=1;
   Query OK, 0 rows affected (0.00 sec)

   mysql> SHOW VARIABLES LIKE
   'rpl_semi_sync_master_enabled';
   +------------------------------+-------+
   ```

```
| Variable_name                | Value |
+------------------------------+-------+
| rpl_semi_sync_master_enabled | ON    |
+------------------------------+-------+
1 row in set (0.00 sec)

mysql> SET @@GLOBAL.rpl_semi_sync_master_timeout=1000;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE
'rpl_semi_sync_master_timeout';
+------------------------------+-------+
| Variable_name                | Value |
+------------------------------+-------+
| rpl_semi_sync_master_timeout | 1000  |
+------------------------------+-------+
1 row in set (0.00 sec)
```

3. On the slave, install the `rpl_semi_sync_slave` plugin:

```
mysql> INSTALL PLUGIN rpl_semi_sync_slave SONAME
'semisync_slave.so';
Query OK, 0 rows affected (0.22 sec)

mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS FROM
INFORMATION_SCHEMA.PLUGINS WHERE PLUGIN_NAME LIKE
'%semi%';
+---------------------+---------------+
| PLUGIN_NAME         | PLUGIN_STATUS |
+---------------------+---------------+
| rpl_semi_sync_slave | ACTIVE        |
+---------------------+---------------+
1 row in set (0.08 sec)
```

4. On the slave, enable semi-synchronous replication and restart the slave IO thread:

```
mysql> SET GLOBAL rpl_semi_sync_slave_enabled = 1;
Query OK, 0 rows affected (0.00 sec)

mysql> STOP SLAVE IO_THREAD;
Query OK, 0 rows affected (0.02 sec)
```

```
mysql> START SLAVE IO_THREAD;
Query OK, 0 rows affected (0.00 sec)
```

5. You can monitor the status of the semi-synchronous replication through the following:
   To find the number of clients connected as semi-sync—on master, execute:

```
mysql> SHOW STATUS LIKE 'Rpl_semi_sync_master_clients';
+------------------------------+-------+
| Variable_name                | Value |
+------------------------------+-------+
| Rpl_semi_sync_master_clients | 1     |
+------------------------------+-------+
1 row in set (0.01 sec)
```

   The master switches between asynchronous and semi-synchronous replication when the timeout occurs and the slaves catch up. To check what type of replication the master is using, check the status of `Rpl_semi_sync_master_status` (on means semi-sync and off means async):

```
mysql> SHOW STATUS LIKE 'Rpl_semi_sync_master_status';
+-----------------------------+-------+
| Variable_name               | Value |
+-----------------------------+-------+
| Rpl_semi_sync_master_status | ON    |
+-----------------------------+-------+
1 row in set (0.00 sec)
```

You can verify the semi-synchronous replication using this method:

1. Stop the slave:

```
mysql> STOP SLAVE;
Query OK, 0 rows affected (0.01 sec)
```

2. On the master, execute any statement:

```
mysql> USE employees;
Database changed

mysql> DROP TABLE IF EXISTS employees_test;
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

You will notice that the master has switched to asynchronous replication since it did not get any acknowledgement from the slave even after 1 second (the value of `rpl_semi_sync_master_timeout`):

```
mysql> SHOW STATUS LIKE 'Rpl_semi_sync_master_status';
+-----------------------------+-------+
| Variable_name               | Value |
+-----------------------------+-------+
| Rpl_semi_sync_master_status | ON    |
+-----------------------------+-------+
1 row in set (0.00 sec)

mysql> DROP TABLE IF EXISTS employees_test;
Query OK, 0 rows affected (1.02 sec)
```

```

mysql> SHOW STATUS LIKE 'Rpl_semi_sync_master_status';
+-----------------------------+-------+
| Variable_name               | Value |
+-----------------------------+-------+
| Rpl_semi_sync_master_status | OFF   |
+-----------------------------+-------+
1 row in set (0.01 sec
```

3. Start the slave:

```
mysql> START SLAVE;
Query OK, 0 rows affected (0.02 sec)
```

4. On the master, you will notice that the master switched back to semi-synchronous replication:

```
mysql> SHOW STATUS LIKE 'Rpl_semi_sync_master_status';
+-----------------------------+-------+
| Variable_name               | Value |
+-----------------------------+-------+
| Rpl_semi_sync_master_status | ON    |
+-----------------------------+-------+
1 row in set (0.00 sec)
```

Table Maintenance

In this chapter, we will cover the following recipes:

- Installing Percona Toolkit
- Altering tables
- Moving tables across databases
- Altering tables using an online schema change tool
- Archiving tables
- Cloning tables
- Partitioning tables
- Partition pruning and selection
- Partition management
- Partition information
- Efficiently managing time to live and soft delete rows

Introduction

One of the key aspects in maintaining a database is managing tables. Often, you need to alter a big table or clone a table. In this chapter, you will learn about managing big tables. Some open source third-party tools are used as MySQL does not support certain operations. The installation and usage of third-party tools are also covered in this chapter.

Installing Percona Toolkit

Percona Toolkit is a collection of advanced open source command-line tools, developed and used by Percona to perform a variety of tasks that are too difficult or complex to perform manually. The installation is covered in this section. In the later sections, you will learn how to use it.

How to do it...

Let us see how to install Percona Toolkit on various operating systems.

On Debian/Ubuntu

1. Download the repository package:

```
shell> wget https://repo.percona.com/apt/percona-
release_0.1-4.$(lsb_release -sc)_all.deb
```

2. Install the repository package:

```
shell> sudo dpkg -i percona-release_0.1-4.$(lsb_release
-sc)_all.deb
```

3. Update local package list:

```
shell> sudo apt-get update
```

4. Make sure that Percona packages are available:

```
shell> apt-cache search percona
```

   You should see output similar to the following:

```
percona-xtrabackup-dbg - Debug symbols for Percona
XtraBackup
percona-xtrabackup-test - Test suite for Percona
XtraBackup
percona-xtradb-cluster-client - Percona XtraDB Cluster
database client
percona-xtradb-cluster-server - Percona XtraDB Cluster
database server
percona-xtradb-cluster-testsuite - Percona XtraDB
Cluster database regression test suite
percona-xtradb-cluster-testsuite-5.5 - Percona Server
database test suite
...
```

5. Install the `percona-toolkit` package:

```
      shell> sudo apt-get install percona-toolkit
```

If you do not want to install a repository, you can also install directly:

```
shell> wget https://www.percona.com/downloads/percona-
toolkit/3.0.4/binary/debian/xenial/x86_64/percona-
toolkit_3.0.4-1.xenial_amd64.deb
```

```
shell> sudo dpkg -i percona-toolkit_3.0.4-
1.yakkety_amd64.deb;
shell> sudo apt-get install -f
```

On CentOS/Red Hat/Fedora

1. Install the repository package:

   ```
   shell> sudo yum install
   http://www.percona.com/downloads/percona-
   release/redhat/0.1-4/percona-release-0.1-4.noarch.rpm
   ```

   You should see the following if successful:

   ```
   Installed:
     percona-release.noarch 0:0.1-4

   Complete!
   ```

2. Make sure that Percona packages are available:

   ```
   shell> sudo yum list | grep percona
   ```

   You should see output similar to the following:

   ```
   percona-release.noarch                     0.1-4
   @/percona-release-0.1-4.noarch
   Percona-Server-55-debuginfo.x86_64         5.5.54-
   rel38.7.el7          percona-release-x86_64
   Percona-Server-56-debuginfo.x86_64         5.6.35-
   rel81.0.el7          percona-release-x86_64
   Percona-Server-57-debuginfo.x86_64         5.7.17-
   13.1.el7             percona-release-x86_64
   ...
   ```

3. Install Percona Toolkit:

   ```
   shell> sudo yum install percona-toolkit
   ```

If you do not want to install a repository, you can directly install using YUM:

```
shell> sudo yum install
https://www.percona.com/downloads/percona-
toolkit/3.0.4/binary/redhat/7/x86_64/percona-toolkit-3.0.4-
1.el7.x86_64.rpm
```

Altering tables

`ALTER TABLE` changes the structure of a table. For example, you can add or delete columns, create or destroy indexes, change the type of existing columns, or rename columns or the table itself.

While performing certain alter operations such as changing a column data type, adding a `SPATIAL INDEX`, dropping a primary key, converting a character set, adding/removing encryption, and so on, DML operations on the table are blocked. If the table is big, it will take even more time to alter, and the application cannot access the table during that time, which is not desired. In those situations, a `pt-online-schema` change is helpful, where DML statements are allowed.

There are two algorithms for alter operations:

- **In-place** (default): Does not require copying whole table data
- **Copy**: Copies the data into a temporary disk file and renames it

Only certain alter operations can be done in-place. The performance of an online DDL operation is largely determined by whether the operation is performed in-place, or requires copying and rebuilding the entire table. Refer to https://dev.mysql.com/doc/refman/8.0/en/innodb-create-index-overview.html#innodb-online-ddl-summary-grid to see what kinds of operations can be performed in-place, and any requirements for avoiding table-copy operations.

How copy algorithm works (taken from the reference manual—https://dev.mysql.com/doc/refman/8.0/en/alter-table.html)

`ALTER TABLE` operations that are not performed *in-place* make a temporary copy of the original table. MySQL waits for other operations that are modifying the table, then proceeds. It incorporates the alteration into the copy, deletes the original table, and renames the new one. While `ALTER TABLE` is executing, the original table is readable by other sessions. Updates and writes to the table that begin after the `ALTER TABLE` operation begins are stalled until the new table is ready, then are automatically redirected to the new table without any failed updates. The temporary copy of the original table is created in the database directory of the new table. This can differ from the database directory of the original table for `ALTER TABLE` operations that rename the table to a different database.

To get an idea on whether a DDL operation performs in-place or a table copy, look at the `rows affected` value displayed after the command finishes:

- Changing the default value of a column (superfast, does not affect the table data at all), the output would be some thing like this:

```
Query OK, 0 rows affected (0.07 sec)
```

- Adding an index (takes time, but `0 rows affected` shows that the table is not copied), the output would be some thing like this:

  ```
  Query OK, 0 rows affected (21.42 sec)
  ```
- Changing the data type of a column (takes substantial time and does require rebuilding all the rows of the table), , the output would be some thing like this:

```
Query OK, 1671168 rows affected (1 min 35.54 sec)
```

Changing the data type of a column requires rebuilding all the rows of the table with the exception of changing the `VARCHAR` size, which may be performed using online `ALTER TABLE`. See the example mentioned in the *Altering tables using an online schema change tool* section, which shows how to modify column properties using `pt-online-schema`.

How to do it...

If you want to add a new column to the `employees` table, you can execute the `ADD COLUMN` statement:

```
mysql> ALTER TABLE employees ADD COLUMN address
varchar(100);
Query OK, 0 rows affected (5.10 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

You can see that the number of rows affected is `0`, which means that the table is not copied and the operation is done in-place.

If you want to increase the length of the `varchar` column, you can execute the `MODIFY COLUMN` statement:

```
mysql> ALTER TABLE employees MODIFY COLUMN address
VARCHAR(255);
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

If you think that `varchar(255)` is not sufficient to store addresses, and you would like to change it to `tinytext`, you can use the `MODIFY COLUMN` statement. However, in this case, since you are modifying the data type of a column, all the rows of the existing table should be modified, which requires table copy, and DMLs are blocked:

```
mysql> ALTER TABLE employees MODIFY COLUMN address tinytext;
Query OK, 300025 rows affected (4.36 sec)
Records: 300025  Duplicates: 0  Warnings: 0
```

You will notice that the rows affected are `300025`, which is the size of the table.

There are various other operations that you can do, such as renaming a column, changing the default value, reordering the column positions, and so on; refer to the manual at https://dev.mysql.com/doc/refman/8.0/en/innodb-create-index-overview.html for more details.

Adding a virtual generated column is just a metadata change and is almost instantaneous:

```
mysql> ALTER TABLE employees ADD COLUMN full_name
VARCHAR(40) AS (CONCAT('first_name', ' ', 'last_name'));
Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

However, adding a `STORED GENERATED` column and modifying the `VIRTUAL GENERATED` column is not online:

```
mysql> ALTER TABLE employees MODIFY COLUMN full_name
VARCHAR(40) AS (CONCAT(first_name, '-', last_name)) VIRTUAL;
Query OK, 300026 rows affected (4.37 sec)
Records: 300026  Duplicates: 0  Warnings: 0
```

Moving tables across databases

You can rename a table by executing the `RENAME TABLE` statement.

For the following illustrations to work, create sample tables and databases

```
mysql> CREATE DATABASE prod;
mysql> CREATE TABLE prod.audit_log (id int NOT NULL, msg
varchar(64));
mysql> CREATE DATABASE archive;
```

How to do it...

For example, if you want to rename the `audit_log` table `audit_log_archive_2018`, you can execute the following:

```
mysql> USE prod;
Database changed

mysql> RENAME TABLE audit_log TO audit_log_archive_2018;
Query OK, 0 rows affected (0.07 sec)
```

If you want to move the table from one database to an other, you can use dot notation to specify the database name. For example, if you want to move the `audit_log` table from the database named `prod` to the database named `archive`, execute the following:

```
mysql> USE prod
Reading table information for completion of table and column
names
You can turn off this feature to get a quicker startup with
-A

mysql> SHOW TABLES;
+------------------------+
| Tables_in_prod         |
+------------------------+
| audit_log_archive_2018 |
+------------------------+
1 row in set (0.00 sec)

mysql> RENAME TABLE audit_log_archive_2018 TO
archive.audit_log;
Query OK, 0 rows affected (0.03 sec)

mysql> SHOW TABLES;
Empty set (0.00 sec)

mysql> USE archive
Reading table information for completion of table and column
```

```
names
You can turn off this feature to get a quicker startup with
-A

Database changed
mysql> SHOW TABLES;
+-------------------+
| Tables_in_archive |
+-------------------+
| audit_log         |
+-------------------+
1 row in set (0.00 sec)
```

Altering tables using an online schema change tool

In this section, you will learn about Percona's `pt-online-schema-change` (`pt-osc`) tool, which is used to perform `ALTER TABLE` operations without blocking DMLs.

`pt-osc` comes along with Percona Toolkit. Installation of Percona Toolkit has already been described earlier in the chapter.

How it works...

(Taken from https://www.percona.com/doc/percona-toolkit/LATEST/pt-online-schema-change.html.)

`pt-online-schema-change` works by creating an empty copy of the table to alter, modifying it as desired, and then copying rows from the original table into the new table. When the copy is complete, it moves away the original table and replaces it with the new one. By default, it also drops the original table.

The data copy process is performed in small chunks of data, which are varied to attempt to make them execute in a specific amount of time. Any modifications to data in the original tables during the copy will be reflected in the new table, because the tool creates triggers on the original table to update the corresponding rows in the new table. The use of triggers means that the tool will not work if any triggers are already defined on the table.

When the tool finishes copying data into the new table, it uses an atomic `RENAME TABLE` operation to simultaneously rename the original and new tables. After this is complete, the tool drops the original table.

Foreign keys complicate the tool's operation and introduce additional risk. The technique of atomically renaming the original and new tables does not work when foreign keys refer to the table. The tool must update foreign keys to refer to the new table after the schema change is complete. The tool supports two methods for accomplishing this. You can read more about this in the documentation for `--alter-foreign-keys-method`.

How to do it...

Modifying the column data type can be done as follows:

```
shell> pt-online-schema-change
D=employees,t=employees,h=localhost -u root --ask-pass --
alter="MODIFY COLUMN address VARCHAR(100)" --alter-foreign-
keys-method=auto --execute
Enter MySQL password:
No slaves found.  See --recursion-method if host server1 has
slaves.
Not checking slave lag because no slaves were found and --
check-slave-lag was not specified.
Operation, tries, wait:
  analyze_table, 10, 1
  copy_rows, 10, 0.25
  create_triggers, 10, 1
  drop_triggers, 10, 1
  swap_tables, 10, 1
  update_foreign_keys, 10, 1
Child tables:
  `employees`.`dept_emp` (approx. 331143 rows)
  `employees`.`titles` (approx. 442605 rows)
  `employees`.`salaries` (approx. 2838426 rows)
  `employees`.`dept_manager` (approx. 24 rows)
Will automatically choose the method to update foreign keys.
Altering `employees`.`employees`...
Creating new table...
Created new table employees._employees_new OK.
Altering new table...
Altered `employees`.`_employees_new` OK.
2017-09-24T09:56:49 Creating triggers...
2017-09-24T09:56:49 Created triggers OK.
2017-09-24T09:56:49 Copying approximately 299478 rows...
2017-09-24T09:56:56 Copied rows OK.
2017-09-24T09:56:56 Max rows for the rebuild_constraints
method: 88074
Determining the method to update foreign keys...
2017-09-24T09:56:56   `employees`.`dept_emp`: too many rows:
331143; must use drop_swap
2017-09-24T09:56:56 Drop-swapping tables...
```

```
2017-09-24T09:56:56 Analyzing new table...
2017-09-24T09:56:56 Dropped and swapped tables OK.
Not dropping old table because --no-drop-old-table was
specified.
2017-09-24T09:56:56 Dropping triggers...
2017-09-24T09:56:56 Dropped triggers OK.
Successfully altered `employees`.`employees`.
```

You will have noticed that the tool has created a new table with a modified structure, created triggers on the table, copied the rows to a new table, and finally, renamed the new table.

If you want to alter the `salaries` table, which already has triggers, you need to specify the `--preserver-triggers` option, otherwise you will end up with the error: `The table `employees`.`salaries` has triggers but --preserve-triggers was not specified.`:

```
shell> pt-online-schema-change
D=employees,t=salaries,h=localhost -u user --ask-pass --
alter="MODIFY COLUMN salary int" --alter-foreign-keys-
method=auto --execute --no-drop-old-table --preserve-
triggers
No slaves found.  See --recursion-method if host server1 has
slaves.
Not checking slave lag because no slaves were found and --
check-slave-lag was not specified.

Operation, tries, wait:
  analyze_table, 10, 1
  copy_rows, 10, 0.25
  create_triggers, 10, 1
  drop_triggers, 10, 1
  swap_tables, 10, 1
  update_foreign_keys, 10, 1
No foreign keys reference `employees`.`salaries`; ignoring -
-alter-foreign-keys-method.
Altering `employees`.`salaries`...
Creating new table...
Created new table employees._salaries_new OK.
Altering new table...
Altered `employees`.`_salaries_new` OK.
```

```
2017-09-24T11:11:58 Creating triggers...
2017-09-24T11:11:58 Created triggers OK.
2017-09-24T11:11:58 Copying approximately 2838045 rows...
2017-09-24T11:12:20 Copied rows OK.
2017-09-24T11:12:20 Adding original triggers to new table.
2017-09-24T11:12:21 Analyzing new table...
2017-09-24T11:12:21 Swapping tables...
2017-09-24T11:12:21 Swapped original and new tables OK.
Not dropping old table because --no-drop-old-table was
specified.
2017-09-24T11:12:21 Dropping triggers...
2017-09-24T11:12:21 Dropped triggers OK.
Successfully altered `employees`.`salaries`
```

If the server has slaves, this tool can create a slave lag while copying from an existing table to a new table. To avoid this, you can specify `--check-slave-lag` (enabled by default); it pauses the data copy until this replica's lag is less than `--max-lag`, which is 1 second by default. You can specify `--max-lag` by passing the `--max-lag` option.

If you want to make sure that the slaves will not be lagged by more than 10 seconds, pass `--max-lag=10`:

```
shell> pt-online-schema-change
D=employees,t=employees,h=localhost -u user --ask-pass --
alter="MODIFY COLUMN address VARCHAR(100)" --alter-foreign-
keys-method=auto --execute --preserve-triggers --max-lag=10
Enter MySQL password:
Found 1 slaves:
server2 -> xx.xxx.xxx.xx:socket
Will check slave lag on:
server2 -> xx.xxx.xxx.xx:socket
Operation, tries, wait:
  analyze_table, 10, 1
  copy_rows, 10, 0.25
  create_triggers, 10, 1
  drop_triggers, 10, 1
  swap_tables, 10, 1
  update_foreign_keys, 10, 1
Child tables:
  `employees`.`dept_emp` (approx. 331143 rows)
  `employees`.`titles` (approx. 442605 rows)
  `employees`.`salaries` (approx. 2838426 rows)
  `employees`.`dept_manager` (approx. 24 rows)
Will automatically choose the method to update foreign keys.
Altering `employees`.`employees`...
Creating new table...
Created new table employees._employees_new OK.
Waiting forever for new table `employees`.`_employees_new`
to replicate to ubuntu...
Altering new table...
Altered `employees`.`_employees_new` OK.
2017-09-24T12:00:58 Creating triggers...
2017-09-24T12:00:58 Created triggers OK.
2017-09-24T12:00:58 Copying approximately 299342 rows...
2017-09-24T12:01:05 Copied rows OK.
2017-09-24T12:01:05 Max rows for the rebuild_constraints
method: 86446
Determining the method to update foreign keys...
2017-09-24T12:01:05   `employees`.`dept_emp`: too many rows:
331143; must use drop_swap
2017-09-24T12:01:05 Skipping triggers creation since --no-
swap-tables was specified along with --drop-new-table
2017-09-24T12:01:05 Drop-swapping tables...
2017-09-24T12:01:05 Analyzing new table...
2017-09-24T12:01:05 Dropped and swapped tables OK.
Not dropping old table because --no-drop-old-table was
specified.
2017-09-24T12:01:05 Dropping triggers...
2017-09-24T12:01:05 Dropped triggers OK.
Successfully altered `employees`.`employees`.
```

For more details and options, refer to the Percona documentation, at https://www.percona.com/doc/percona-toolkit/LATEST/pt-online-schema-change.html.

`pt-online-schema-change` works only when there is a primary key or unique key, otherwise it will fail with the following error:

```
The new table `employees`.`_employees_new` does
not have a PRIMARY KEY or a unique index which is
```

```
|required for the DELETE trigger.
```

So, if the table does not have any unique key, you cannot use `pt-online-schema-change`.

Archiving tables

Sometimes, you do not want to keep older data and wish to delete it. If you want to delete all the rows which were last accessed over a month ago, if the table is small (<10k rows), you can straight away use the following:

```
DELETE FROM <TABLE> WHERE last_accessed<DATE_ADD(NOW(),
INTERVAL -1 MONTH)
```

What happens if the table is big? You know `InnoDB` creates an `UNDO` log to restore failed transactions. So all the deleted rows are saved in the `UNDO` log space to be used to restore in case the `DELETE` statement aborts in between. Unfortunately, if the `DELETE` statement is aborted in between, `InnoDB` copies the rows from the `UNDO` log space to table, which can make the table inaccessible.

To overcome this behavior, you can `LIMIT` the number of rows deleted and `COMMIT` the transaction, running the same thing in a loop until all the unwanted rows are deleted.

This is an example pseudo code:

```
WHILE count<=0:
    DELETE FROM <TABLE> WHERE last_accessed<DATE_ADD(NOW(),
INTERVAL -1 MONTH) LIMIT 10000;
    count=SELECT COUNT(*) FROM <TABLE> WHERE
last_accessed<DATE_ADD(NOW(), INTERVAL -1 MONTH);
```

If there is no `INDEX` on `last_accessed`, it can lock the table. In that case, you need to find out the primary key for the deleted rows and delete based on `PRIMARY KEY`.

This is the pseudo code, assuming `id` is the `PRIMARY KEY`:

```
WHILE count<=0:
    SELECT id FROM <TABLE> WHERE last_accessed <
DATE_ADD(NOW(), INTERVAL -1 MONTH) LIMIT 10000;
    DELETE FROM <TABLE> WHERE id IN ('ids from above
statement');
    count=SELECT COUNT(*) FROM <TABLE> WHERE
last_accessed<DATE_ADD(NOW(), INTERVAL -1 MONTH);
```

Instead of writing the code for deleting the rows, you can use Percona's `pt-archiver` tool, which essentially does the same and provides many other options, such as saving the rows into another table or file, fine control over the load and replication delay, and so on.

How to do it...

There are many options in `pt-archiver`, we will start with simple purging.

Purging data

If you want to delete all the rows from the `employees` table for which `hire_date` is older than 30 years, you can execute the following:

```
shell> pt-archiver --source
h=localhost,D=employees,t=employees -u <user> -p<pass> --
where="hire_date<DATE_ADD(NOW(), INTERVAL -30 YEAR)" --no-
check-charset --limit 10000 --commit-each
```

You can pass the hostname, database name, and table name through the `--source` option. You can limit the number of rows to delete in a batch using the `--limit` option.

If you specify `--progress`, the output is a header row, plus status output at intervals. Each row in the status output lists the current date and time, how many seconds `pt-archiver` has been running, and how many rows it has archived.

If you specify `--statistics`, `pt-archiver` outputs timing and other information to help you identify which part of your archiving process takes the most time.

If you specify `--check-slave-lag`, the tool will pause archiving until the slave lag is less than `--max-lag`.

Archiving data

If you want to save the rows after deletion into a separate table or file, you can specify the `--dest` option.

Suppose you want to move all the rows of the `employees` table from the `employees` database to the `employees_archive` table, you can execute the following:

```
shell> pt-archiver --source
h=localhost,D=employees,t=employees --dest
h=localhost,D=employees_archive -u <user> -p<pass> --
where="1=1" --no-check-charset --limit 10000 --commit-each
```

If you specify `--where="1=1"`, it copies all rows.

Copying data

If you want to copy data from one table to another, you can either use `mysqldump` or `mysqlpump` to back up certain rows and then load them into the destination table. As an alternative, you can also use `pt-archive`. If you specify the `--no-delete` option, `pt-archiver` will not delete the rows from the source:

```
shell> pt-archiver --source
h=localhost,D=employees,t=employees --dest
h=localhost,D=employees_archive -u <user> -p<pass> --
where="1=1" --no-check-charset --limit 10000 --commit-each -
-no-delete
```

See also

Refer to https://www.percona.com/doc/percona-toolkit/LATEST/pt-archiver.html for more details about and options for `pt-archiver`.

Cloning tables

If you want to clone a table, there are many options.

How to do it...

1. Use the `INSERT INTO SELECT` statement:

```
      mysql> CREATE TABLE employees_clone LIKE employees;
      mysql> INSERT INTO employees_clone SELECT * FROM
      employees;
```

    Note that if there are any generated columns, the above statement would not work. In that case, you should give full insert statement excluding the generated columns.

```
mysql> INSERT INTO employees_clone SELECT * FROM employees;
ERROR 3105 (HY000): The value specified for generated column
'hire_date_year' in table 'employees_clone' is not allowed.

mysql> INSERT INTO employees_clone(emp_no, birth_date,
first_name, last_name, gender, hire_date) SELECT emp_no,
birth_date, first_name, last_name, gender, hire_date FROM
employees;
Query OK, 300024 rows affected (3.21 sec)
Records: 300024  Duplicates: 0  Warnings: 0
```

But the preceding statement is very slow and dangerous on big tables. Remember, if the statement fails, to restore the table state, `InnoDB` saves all the rows in `UNDO` logs.

2. Use `mysqldump` or `mysqlpump` and take a backup of a single table and restore it on destination. This can take very long time if the table is big.
3. Use `Innobackupex` to take a backup of specific table and restore the data file onto destination.
4. Use `pt-archiver` with the `--no-delete` option, which will copy the desired rows or all rows to the destination table.

You can also clone tables by using transportable tablespaces, which are explained in the *Copying file-per-table tablespaces to another instance* section of Chapter 11, *Managing Tablespace.*

Partitioning tables

You can distribute portions of individual tables across a filesystem using partitions. The user-selected rule by which the division of data is accomplished is known as a partitioning function, which can be modulus, simple matching against a set of ranges or value lists, an internal hashing function, or a linear hashing function.

Different rows of a table may be assigned to different physical partitions, which is called horizontal partitioning. MySQL does not have support for vertical partitioning, in which different columns of a table are assigned to different physical partitions.

There are many ways to partition a table:

- `RANGE`: This type of partitioning assigns rows to partitions based on column values falling within a given range.
- `LIST`: Similar to partitioning by `RANGE`, except that the partition is selected based on columns matching one of a set of discrete values.
- `HASH`: With this type of partitioning, a partition is selected based on the value returned by a user-defined expression that operates on column values in rows to be inserted into the table. The function may consist of any expression valid in MySQL that yields a non-negative integer value.
- `KEY`: This type of partitioning is similar to partitioning by `HASH`, except that only one or more columns to be evaluated are supplied, and the MySQL server provides its own hashing function. These columns can contain other than integer values,

since the hashing function supplied by MySQL guarantees an integer result regardless of the column data type.

Each of the preceding partition types have an extension. `RANGE` has `RANGE COLUMNS`, `LIST` has `LIST COLUMNS`, `HASH` has `LINEAR HASH`, and `KEY` has `LINEAR KEY`.

For `[LINEAR] KEY`, `RANGE COLUMNS`, and `LIST COLUMNS` partitioning, the partitioning expression consists of a list of one or more columns.

In the case of `RANGE`, `LIST`, and `[LINEAR] HASH` partitioning, the value of the partitioning column is passed to the partitioning function, which returns an integer value representing the number of the partition in which that particular record should be stored. This function must be non-constant and non-random.

A very common use of database partitioning is to segregate data by date.

Refer to https://dev.mysql.com/doc/refman/8.0/en/partitioning-overview.html for advantages and other details on partitioning.

Note that partitions work only for `InnoDB` tables, and foreign keys are not yet supported in conjunction with partitioning.

How to do it...

You can specify the partitioning while creating the table or by executing the `ALTER TABLE` command. The partition column should be part of all the unique keys in the table.

If you defined partitions based on the `created_at` column and `id` is the primary key, you should include the `create_at` column as part of `PRIMARY KEY`, that is, (`id`, `created_at`).

The following example assumes that there are no foreign keys referenced to the table.

If you wish to implement a partitioning scheme based on ranges or intervals of time in MySQL 8.0, you have two options:

- Partition the table by `RANGE`, and for the partitioning expression, employ a function operating on a `DATE`, `TIME`, or `DATETIME` column and returning an integer value
- Partition the table by `RANGE COLUMNS`, using a `DATE` or `DATETIME` column as the partitioning column

RANGE partitioning

If you want to partition the `employees` table based on `emp_no` and you want to keep 100,000 employees in one partition, you can create it like this:

```
mysql> CREATE TABLE `employees` (
  `emp_no` int(11) NOT NULL,
  `birth_date` date NOT NULL,
  `first_name` varchar(14) NOT NULL,
  `last_name` varchar(16) NOT NULL,
  `gender` enum('M','F') NOT NULL,
  `hire_date` date NOT NULL,
  `address` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`emp_no`),
  KEY `name` (`first_name`,`last_name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
PARTITION BY RANGE (emp_no)
(PARTITION p0 VALUES LESS THAN (100000) ENGINE = InnoDB,
 PARTITION p1 VALUES LESS THAN (200000) ENGINE = InnoDB,
 PARTITION p2 VALUES LESS THAN (300000) ENGINE = InnoDB,
 PARTITION p3 VALUES LESS THAN (400000) ENGINE = InnoDB,
 PARTITION p4 VALUES LESS THAN (500000) ENGINE = InnoDB);
```

So, all the employees having `emp_no` less than 100,000 will go to partition `p0` and all the employees having the `emp_no` less than `200000` and greater than `100000` will go to partition `p1`, and so on.

If the employee number is above `500000`, since there is no partition defined for them, the insert will fail with error. To avoid this, you have to regularly check and add partitions or create a `MAXVALUE` partition to catch all such exceptions:

```
mysql> CREATE TABLE `employees` (
  `emp_no` int(11) NOT NULL,
  `birth_date` date NOT NULL,
  `first_name` varchar(14) NOT NULL,
  `last_name` varchar(16) NOT NULL,
  `gender` enum('M','F') NOT NULL,
  `hire_date` date NOT NULL,
  `address` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`emp_no`),
  KEY `name` (`first_name`,`last_name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
PARTITION BY RANGE (emp_no)
(PARTITION p0 VALUES LESS THAN (100000) ENGINE = InnoDB,
 PARTITION p1 VALUES LESS THAN (200000) ENGINE = InnoDB,
 PARTITION p2 VALUES LESS THAN (300000) ENGINE = InnoDB,
 PARTITION p3 VALUES LESS THAN (400000) ENGINE = InnoDB,
 PARTITION p4 VALUES LESS THAN (500000) ENGINE = InnoDB,
 PARTITION pmax VALUES LESS THAN MAXVALUE ENGINE = InnoDB
);
```

If you want to partition based on `hire_date`, you can use the `YEAR(hire_date)` function as the partition expression:

```
mysql> CREATE TABLE `employees` (
  `emp_no` int(11) NOT NULL,
  `birth_date` date NOT NULL,
  `first_name` varchar(14) NOT NULL,
  `last_name` varchar(16) NOT NULL,
  `gender` enum('M','F') NOT NULL,
  `hire_date` date NOT NULL,
  `address` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`emp_no`,`hire_date`),
  KEY `name` (`first_name`,`last_name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
PARTITION BY RANGE (YEAR(hire_date))
(PARTITION p1980 VALUES LESS THAN (1980) ENGINE = InnoDB,
 PARTITION p1990 VALUES LESS THAN (1990) ENGINE = InnoDB,
 PARTITION p2000 VALUES LESS THAN (2000) ENGINE = InnoDB,
 PARTITION p2010 VALUES LESS THAN (2010) ENGINE = InnoDB,
 PARTITION p2020 VALUES LESS THAN (2020) ENGINE = InnoDB,
 PARTITION pmax VALUES LESS THAN MAXVALUE ENGINE = InnoDB
);
```

Partitioning in MySQL is widely used on `date`, `datetime`, or `timestamp` columns. If you want to store some events on the database, and all the queries are based out of a time range, you can use partitioning like this.

The partitioning function `to_days()` returns the number of days since `0000-01-01`, which is a integer:

```
mysql> CREATE TABLE `event_history` (
  `event_id` int(11) NOT NULL,
  `event_name` varchar(10) NOT NULL,
  `created_at` datetime NOT NULL,
  `last_updated` timestamp DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP,
  `event_type` varchar(10) NOT NULL,
  `msg` tinytext NOT NULL,
  PRIMARY KEY (`event_id`,`created_at`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
PARTITION BY RANGE (to_days(created_at))
(PARTITION p20170930 VALUES LESS THAN (736967) ENGINE =
InnoDB,
PARTITION p20171001 VALUES LESS THAN (736968) ENGINE =
InnoDB,
PARTITION p20171002 VALUES LESS THAN (736969) ENGINE =
InnoDB,
PARTITION p20171003 VALUES LESS THAN (736970) ENGINE =
InnoDB,
PARTITION p20171004 VALUES LESS THAN (736971) ENGINE =
InnoDB,
PARTITION p20171005 VALUES LESS THAN (736972) ENGINE =
InnoDB,
PARTITION p20171006 VALUES LESS THAN (736973) ENGINE =
InnoDB,
PARTITION p20171007 VALUES LESS THAN (736974) ENGINE =
InnoDB,
PARTITION p20171008 VALUES LESS THAN (736975) ENGINE =
InnoDB,
PARTITION p20171009 VALUES LESS THAN (736976) ENGINE =
InnoDB,
PARTITION p20171010 VALUES LESS THAN (736977) ENGINE =
InnoDB,
PARTITION p20171011 VALUES LESS THAN (736978) ENGINE =
InnoDB,
PARTITION p20171012 VALUES LESS THAN (736979) ENGINE =
InnoDB,
PARTITION p20171013 VALUES LESS THAN (736980) ENGINE =
InnoDB,
PARTITION p20171014 VALUES LESS THAN (736981) ENGINE =
InnoDB,
PARTITION p20171015 VALUES LESS THAN (736982) ENGINE =
InnoDB,
PARTITION pmax VALUES LESS THAN MAXVALUE ENGINE = InnoDB
);
```

If you want to convert an existing table into a partitioned table, and if the partition key is not part of `PRIMARY KEY`, you need to drop `PRIMARY KEY` and add the partition key as part of `PRIMARY KEY` and all unique keys. Otherwise, you will get the error `ERROR 1503 (HY000): A PRIMARY KEY must include all columns in the table's partitioning function.`. You can do that as follows:

```
mysql> ALTER TABLE employees DROP PRIMARY KEY, ADD PRIMARY
KEY(emp_no,hire_date);
Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> ALTER TABLE employees PARTITION BY RANGE
(YEAR(hire_date))
        (PARTITION p1980 VALUES LESS THAN (1980) ENGINE =
InnoDB,
        PARTITION p1990 VALUES LESS THAN (1990) ENGINE =
InnoDB,
        PARTITION p2000 VALUES LESS THAN (2000) ENGINE =
InnoDB,
        PARTITION p2010 VALUES LESS THAN (2010) ENGINE =
InnoDB,
        PARTITION p2020 VALUES LESS THAN (2020) ENGINE =
InnoDB,
        PARTITION pmax VALUES LESS THAN MAXVALUE ENGINE =
InnoDB
       );
Query OK, 300025 rows affected (4.71 sec)
Records: 300025  Duplicates: 0  Warnings: 0
```

For more details on `RANGE` partitioning, refer to https://dev.mysql.com/doc/refman/8.0/en/partitioning-range.html.

Removing partitioning

If you wish to remove partitioning, you can execute the `REMOVE PARTITIONING` statement:

```
mysql> ALTER TABLE employees REMOVE PARTITIONING;
Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

RANGE COLUMNS partitioning

`RANGE COLUMNS` partitioning is similar to `RANGE` partitioning, but enables you to define partitions using ranges based on multiple column values. In addition, you can define the ranges using columns of types other than integer types. `RANGE COLUMNS` partitioning differs significantly from `RANGE` partitioning in the following ways:

- `RANGE COLUMNS` does not accept expressions, only names of columns
- `RANGE COLUMNS` accepts a list of one or more columns
- `RANGE COLUMNS` partitioning columns are not restricted to integer columns; string, `DATE` and `DATETIME` columns can also be used as partitioning columns

Instead of using the `to_days()` or `year()` function, you can directly use the column `hire_date` in `RANGE COLUMNS`:

```
mysql> ALTER TABLE employees
    PARTITION BY RANGE COLUMNS (hire_date)
    (PARTITION p0 VALUES LESS THAN ('1970-01-01'),
     PARTITION p1 VALUES LESS THAN ('1980-01-01'),
     PARTITION p2 VALUES LESS THAN ('1990-01-01'),
     PARTITION p3 VALUES LESS THAN ('2000-01-01'),
     PARTITION p4 VALUES LESS THAN ('2010-01-01'),
     PARTITION p5 VALUES LESS THAN (MAXVALUE)
    );
Query OK, 300025 rows affected (4.71 sec)
Records: 300025  Duplicates: 0  Warnings: 0
```

Or you can divide employees based on their `last_name`. This will not guarantee uniform distribution across the partitions:

```
mysql> ALTER TABLE employees
PARTITION BY RANGE COLUMNS (last_name)
    (PARTITION p0 VALUES LESS THAN ('b'),
     PARTITION p1 VALUES LESS THAN ('f'),
     PARTITION p2 VALUES LESS THAN ('l'),
     PARTITION p3 VALUES LESS THAN ('q'),
     PARTITION p4 VALUES LESS THAN ('u'),
     PARTITION p5 VALUES LESS THAN ('z')
  );
Query OK, 300025 rows affected (4.71 sec)
Records: 300025  Duplicates: 0  Warnings: 0
```

Using `RANGE COLUMNS`, you can put multiple columns in the partition function:

```
mysql> CREATE TABLE range_columns_example (
    a INT,
    b INT,
    c INT,
    d INT,
    e INT,
    PRIMARY KEY(a, b, c)
)
PARTITION BY RANGE COLUMNS(a,b,c) (
    PARTITION p0 VALUES LESS THAN (0,25,50),
    PARTITION p1 VALUES LESS THAN (10,50,100),
    PARTITION p2 VALUES LESS THAN (10,100,200),
    PARTITION p3 VALUES LESS THAN
(MAXVALUE,MAXVALUE,MAXVALUE)
 );
```

If you insert values `a=10`, `b=20`, `c=100`, `d=100`, `e=100`, it goes to `p1`. When designing tables partitioned by `RANGE COLUMNS`, you can always test successive partition definitions by comparing the desired tuples using the `mysql` client, like this:

```
mysql> SELECT (10,20,100) < (0,25,50) p0, (10,20,100) <
(10,50,100) p1, (10,20,100) < (10,100,200) p2;
+----+----+----+
| p0 | p1 | p2 |
+----+----+----+
|  0 |  1 |  1 |
+----+----+----+
1 row in set (0.00 sec)
```

In this case, the insert goes to `p1`.

LIST and LIST COLUMNS partitioning

`LIST` partitioning is similar to `RANGE` partitioning, where each partition is defined and selected based on the membership of a column value in one of a set of value lists, rather than in one of a set of contiguous ranges of values.

You need to define it by `PARTITION BY LIST(<expr>)`, where `expr` is a column value or an expression based on a column value and returning an integer value.

Partition definition contains `VALUES IN (<value_list>)`, where `value_list` is a comma-separated list of integers rather than `VALUES LESS THAN (<value>)`.

If you wish to use data types other than integers, you can use `LIST COLUMNS`.

Unlike the case with `RANGE` partitioning, there is no `catch-all` such as `MAXVALUE`; all expected values for the partitioning expression should be covered in the `PARTITION` expression.

Suppose there is a customer table with a zip code and city. If, for example, you want to divide the customers with certain zip codes in a partition, you can use `LIST` partitions:

```
mysql> CREATE TABLE customer (
customer_id INT,
zipcode INT,
city varchar(100),
PRIMARY KEY (customer_id, zipcode)
)
PARTITION BY LIST(zipcode) (
   PARTITION pnorth VALUES IN (560030, 560007, 560051,
560084),
   PARTITION peast VALUES IN (560040, 560008, 560061,
560085),
   PARTITION pwest VALUES IN (560050, 560009, 560062,
560086),
   PARTITION pcentral VALUES IN (560060, 560010, 560063,
560087)
);
```

If you wish to use the columns directly, rather than integers, you can use `LIST COLUMNS`:

```
mysql> CREATE TABLE customer (
customer_id INT,
zipcode INT,
city varchar(100),
PRIMARY KEY (customer_id, city)
)
PARTITION BY LIST COLUMNS(city) (
   PARTITION pnorth VALUES IN ('city1','city2','city3'),
   PARTITION peast VALUES IN ('city4','city5','city6'),
   PARTITION pwest VALUES IN ('city7','city8','city9'),
   PARTITION pcentral VALUES IN ('city10','city11','city12')
);
```

HASH and LINEAR HASH partitioning

Partitioning by `HASH` is used primarily to ensure an even distribution of data among a predetermined number of partitions. With range or list partitioning, you must specify explicitly which partition a given column value or set of column values should be stored in; with hash partitioning, this decision is taken care of for you, and you need only specify a column value or expression based on a column value to be hashed and the number of partitions into which the partitioned table is to be divided.

If you want to evenly distribute employees, instead of `RANGE` partitioning over `YEAR(hire_date)`, you can use `HASH of YEAR(hire_date)` and specify the number of partitions. When `PARTITION BY HASH` is used, the storage engine determines which partition to use based on the modulus of the result of the expression.

For example, if the `hire_date` is `1987-11-28`, `YEAR(hire_date)` would be `1987` and `MOD(1987,8)` is `3`. So the row goes to third partition:

```
mysql> CREATE TABLE `employees` (
  `emp_no` int(11) NOT NULL,
  `birth_date` date NOT NULL,
  `first_name` varchar(14) NOT NULL,
  `last_name` varchar(16) NOT NULL,
  `gender` enum('M','F') NOT NULL,
  `hire_date` date NOT NULL,
  `address` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`emp_no`,`hire_date`),
  KEY `name` (`first_name`,`last_name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```

```
PARTITION BY HASH(YEAR(hire_date))
PARTITIONS 8;
```

The most efficient hashing function is one which operates upon a single table column and whose value increases or decreases consistently with the column value.

In `LINEAR HASH` partitioning, you can use the same syntax, except for adding a `LINEAR` keyword. Instead of a `MODULUS` operation, MySQL uses a powers-of-two algorithm for determining the partition. Refer to https://dev.mysql.com/doc/refman/8.0/en/partitioning-linear-hash.html for more details:

```
mysql> CREATE TABLE `employees` (
  `emp_no` int(11) NOT NULL,
  `birth_date` date NOT NULL,
  `first_name` varchar(14) NOT NULL,
  `last_name` varchar(16) NOT NULL,
  `gender` enum('M','F') NOT NULL,
  `hire_date` date NOT NULL,
  `address` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`emp_no`,`hire_date`),
  KEY `name` (`first_name`,`last_name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
PARTITION BY LINEAR HASH(YEAR(hire_date))
PARTITIONS 8;
```

KEY and LINEAR KEY partitioning

Partitioning by key is similar to partitioning by hash, except that, where hash partitioning employs a user-defined expression, the hashing function for key partitioning is supplied by the MySQL server. This internal hashing function is based on the same algorithm as the `PASSWORD()` function.

`KEY` takes only a list of zero or more column names. Any columns used as the partitioning key must comprise part or all of the table's primary key, if the table has one. Where no column name is specified as the partitioning key, the table's primary key is used, if there is one:

```
mysql> CREATE TABLE `employees` (
  `emp_no` int(11) NOT NULL,
  `birth_date` date NOT NULL,
  `first_name` varchar(14) NOT NULL,
  `last_name` varchar(16) NOT NULL,
  `gender` enum('M','F') NOT NULL,
  `hire_date` date NOT NULL,
  `address` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`emp_no`,`hire_date`),
  KEY `name` (`first_name`,`last_name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
PARTITION BY KEY()
PARTITIONS 8;
```

Subpartitioning

You can further divide each partition into a partitioned table. This is called **subpartitioning** or **composite partitioning**:

```
mysql> CREATE TABLE `employees` (
  `emp_no` int(11) NOT NULL,
  `birth_date` date NOT NULL,
  `first_name` varchar(14) NOT NULL,
  `last_name` varchar(16) NOT NULL,
  `gender` enum('M','F') NOT NULL,
  `hire_date` date NOT NULL,
  `address` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`emp_no`,`hire_date`),
  KEY `name` (`first_name`,`last_name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
PARTITION BY RANGE( YEAR(hire_date) )
  SUBPARTITION BY HASH(emp_no)
    SUBPARTITIONS 4 (
        PARTITION p0 VALUES LESS THAN (1990),
        PARTITION p1 VALUES LESS THAN (2000),
        PARTITION p2 VALUES LESS THAN (2010),
        PARTITION p3 VALUES LESS THAN (2020),
        PARTITION p4 VALUES LESS THAN MAXVALUE
    );
```

Partition pruning and selection

MySQL does not scan partitions where there are no matching values; this is automatic and is called partition pruning. The MySQL optimizer evaluates the partitioning expression for the value given, determines which partition contains that value, and scans only that partition.

`SELECT`, `DELETE`, and `UPDATE` statements support partition pruning. `INSERT` statements currently cannot be pruned.

You can also explicitly specify partitions and subpartitions for rows matching a given `WHERE` condition.

How to do it...

Partition pruning applies only to queries, but explicit selection of partitions is supported for both queries and a number of DML statements.

Partition pruning

Take the example of the `employees` table, which is partitioned based on `emp_no`:

```
mysql> CREATE TABLE `employees` (
  `emp_no` int(11) NOT NULL,
  `birth_date` date NOT NULL,
  `first_name` varchar(14) NOT NULL,
  `last_name` varchar(16) NOT NULL,
  `gender` enum('M','F') NOT NULL,
  `hire_date` date NOT NULL,
  `address` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`emp_no`,`hire_date`),
  KEY `name` (`first_name`,`last_name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
PARTITION BY RANGE (YEAR(hire_date))
(PARTITION p1980 VALUES LESS THAN (1980) ENGINE = InnoDB,
 PARTITION p1990 VALUES LESS THAN (1990) ENGINE = InnoDB,
 PARTITION p2000 VALUES LESS THAN (2000) ENGINE = InnoDB,
 PARTITION p2010 VALUES LESS THAN (2010) ENGINE = InnoDB,
 PARTITION p2020 VALUES LESS THAN (2020) ENGINE = InnoDB,
 PARTITION pmax VALUES LESS THAN MAXVALUE ENGINE = InnoDB
);
```

Suppose the following `SELECT` query is executed:

```
mysql> SELECT last_name,birth_date FROM employees WHERE
hire_date='1999-02-01' AND first_name='Mariangiola';
```

MySQL optimizer detects that partitioned column is being used in the query and automatically determines the partition to scan.

In this query, it first calculates the `YEAR('1999-02-01')`, which is `1999`, and scans the `p2000` partition rather than the whole table. This dramatically reduces the query time.

Instead of `hire_date='1999-02-01'`, if a range is given, such as `hire_date>='1999-02-01'`, then the partitions `p2000`, `p2010`, `p2020`, and `pmax` are scanned.

If the expression `hire_date='1999-02-01'` is not given in the `WHERE` clause, MySQL has to scan the whole table.

To know which partitions the optimizer scans, you can execute the `EXPLAIN` plan of the query, which is explained in the *Explain plan* section of Chapter 13, *Performance Tuning*:

```
mysql> EXPLAIN SELECT last_name,birth_date FROM employees
WHERE hire_date='1999-02-01' AND first_name='Mariangiola'\G
*************************** 1. row
***************************
           id: 1
  select_type: SIMPLE
        table: employees
   partitions: p2000
         type: ref
possible_keys: name
          key: name
      key_len: 58
          ref: const
         rows: 120
     filtered: 10.00
        Extra: Using index condition
```

```
mysql> EXPLAIN SELECT last_name,birth_date FROM employees
WHERE hire_date>='1999-02-01' AND first_name='Mariangiola'\G
*************************** 1. row
***************************
           id: 1
  select_type: SIMPLE
        table: employees
   partitions: p2000,p2010,p2020,pmax
         type: ref
possible_keys: name
          key: name
      key_len: 58
```

```
          ref: const
         rows: 121
     filtered: 33.33
        Extra: Using index condition
1 row in set, 1 warning (0.00 sec)
```

Partition selection

Partition pruning is automatic selection based on the `WHERE` clause. You can explicitly specify the partitions to scan in the query. The queries can be `SELECT`, `DELETE`, `INSERT`, `REPLACE`, `UPDATE`, `LOAD DATA`, and `LOAD XML`. The `PARTITION` option is used to select partitions from a given table, you should specify the keyword `PARTITION` <partition name> immediately following the name of the table, before all other options, including any table alias, for example:

```
mysql> SELECT emp_no,hire_date FROM employees PARTITION
(p1990) LIMIT 10;
+--------+------------+
| emp_no | hire_date  |
+--------+------------+
| 413688 | 1989-12-10 |
| 242368 | 1989-08-06 |
| 283280 | 1985-11-22 |
| 405098 | 1985-11-16 |
|  30404 | 1985-07-17 |
| 419259 | 1988-03-21 |
| 466254 | 1986-11-28 |
| 428971 | 1986-12-13 |
|  94467 | 1987-01-28 |
| 259555 | 1987-07-30 |
+--------+------------+
10 rows in set (0.00 sec)
```

Similarly, we can delete:

```
mysql> DELETE FROM employees PARTITION (p1980, p1990) WHERE
first_name LIKE 'j%';
Query OK, 7001 rows affected (0.12 sec)
```

Partition management

The most important thing when it comes to managing partitions is adding sufficient partitions in advance for time-based `RANGE` partitioning. Failure to do so leads to errors while inserting or, if the `MAXVALUE` partition is defined, all the inserts go into the `MAXVALUE` partition. For example, take the `event_history` table without the `pmax` partition:

```
mysql> CREATE TABLE `event_history` (
  `event_id` int(11) NOT NULL,
  `event_name` date NOT NULL,
  `created_at` datetime NOT NULL,
  `last_updated` timestamp DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP,
  `event_type` varchar(10) NOT NULL,
  `msg` tinytext NOT NULL,
  PRIMARY KEY (`event_id`,`created_at`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
PARTITION BY RANGE (to_days(created_at))
(PARTITION p20170930 VALUES LESS THAN (736967) ENGINE =
InnoDB,
PARTITION p20171001 VALUES LESS THAN (736968) ENGINE =
InnoDB,
PARTITION p20171002 VALUES LESS THAN (736969) ENGINE =
InnoDB,
PARTITION p20171003 VALUES LESS THAN (736970) ENGINE =
InnoDB,
PARTITION p20171004 VALUES LESS THAN (736971) ENGINE =
InnoDB,
PARTITION p20171005 VALUES LESS THAN (736972) ENGINE =
InnoDB,
PARTITION p20171006 VALUES LESS THAN (736973) ENGINE =
InnoDB,
PARTITION p20171007 VALUES LESS THAN (736974) ENGINE =
InnoDB,
PARTITION p20171008 VALUES LESS THAN (736975) ENGINE =
InnoDB,
PARTITION p20171009 VALUES LESS THAN (736976) ENGINE =
InnoDB,
PARTITION p20171010 VALUES LESS THAN (736977) ENGINE =
InnoDB,
PARTITION p20171011 VALUES LESS THAN (736978) ENGINE =
InnoDB,
PARTITION p20171012 VALUES LESS THAN (736979) ENGINE =
InnoDB,
PARTITION p20171013 VALUES LESS THAN (736980) ENGINE =
InnoDB,
PARTITION p20171014 VALUES LESS THAN (736981) ENGINE =
InnoDB,
PARTITION p20171015 VALUES LESS THAN (736982) ENGINE =
InnoDB
);
```

The table accepts the `INSERTS` till October 15, 2017; after that, the `INSERTS` fail.

Another important thing is to `DELETE` the data after it crosses retention.

How to do it...

To perform these operations, you need to execute the `ALTER` command.

ADD partitions

To add a new partition, execute the `ADD PARTITION (<PARTITION DEFINITION>)` statement:

```
mysql> ALTER TABLE event_history ADD PARTITION (
PARTITION p20171016 VALUES LESS THAN (736983) ENGINE =
InnoDB,
PARTITION p20171017 VALUES LESS THAN (736984) ENGINE =
InnoDB
);
```

This statement locks the whole table for a very short time.

Reorganizing partitions

If the `MAXVALUE` partition is there, you cannot add a partition after `MAXVALUE`; in that case, you need to the `REORGANIZE MAXVALUE` partition into two partitions:

```
mysql> ALTER TABLE event_history REORGANIZE PARTITION pmax
INTO (PARTITION p20171016 VALUES LESS THAN (736983) ENGINE =
InnoDB,
PARTITION pmax VALUES LESS THAN MAXVALUE ENGINE = InnoDB);
```

Remember MySQL has to substantially move the data while reorganizing partitions and the table will be locked during that period.

You can also reorganize multiple partitions into a single partition:

```
mysql> ALTER TABLE event_history REORGANIZE PARTITION
p20171001,p20171002,p20171003,p20171004,p20171005,p20171006,
p20171007
INTO (PARTITION p2017_oct_week1 VALUES LESS THAN (736974));
```

DROP partitions

If the data has crossed the retention, you can `DROP` the whole partition, which is superquick compared with conventional `DELETE FROM TABLE` statement. This is very helpful in archiving the data efficiently.

If `p20170930` has crossed the retention, you can `DROP` the partition using the `ALTER TABLE ... DROP PARTITION` statement:

```
mysql> ALTER TABLE event_history DROP PARTITION p20170930;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Dropping the partition removes the `PARTITION DEFINITION` from the table.

TRUNCATE partitions

If you wish to keep `PARTITION DEFINITION` in the table and remove only the data, you can execute the `TRUNCATE PARTITION` command:

```
mysql> ALTER TABLE event_history TRUNCATE PARTITION
p20171001;
Query OK, 0 rows affected (0.08 sec)
```

Managing HASH and KEY partitions

The operations performed on `HASH` and `KEY` partitions are quite different. You can only reduce or increase the number of partitions.

Suppose the `employees` table is partitioned based on `HASH`:

```
mysql> CREATE TABLE `employees` (
  `emp_no` int(11) NOT NULL,
  `birth_date` date NOT NULL,
  `first_name` varchar(14) NOT NULL,
  `last_name` varchar(16) NOT NULL,
  `gender` enum('M','F') NOT NULL,
  `hire_date` date NOT NULL,
  `address` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`emp_no`,`hire_date`),
  KEY `name` (`first_name`,`last_name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
PARTITION BY HASH(YEAR(hire_date))
PARTITIONS 8;
```

To reduce the partitions from `8` to `6`, you can execute the `COALESCE PARTITION` statement and specify the number of partitions you want to reduce, that is, *8-6=2*:

```
mysql> ALTER TABLE employees COALESCE PARTITION 2;
Query OK, 0 rows affected (0.31 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

To increase the partitions from `6` to `16`, you can execute the `ADD PARTITION` statement and specify the number of partitions you want to increase, that is, *16-6=10*:

```
mysql> ALTER TABLE employees ADD PARTITION PARTITIONS 10;
Query OK, 0 rows affected (5.11 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Other operations

You can also perform other operations, such as `REBUILD`, `OPTIMIZE`, `ANALYZE`, and `REPAIR` statements, for a particular partition, for example:

```
mysql> ALTER TABLE event_history REPAIR PARTITION p20171009,
p20171010;
```

Partition information

This section discusses obtaining information about existing partitions, which can be done in a number of ways.

How to do it...

Let's get into the details.

Using SHOW CREATE TABLE

To know whether a table is partitioned or not, you can execute the `SHOW CREATE TABLE\G` statement, which shows the table definition along with partitions, for example:

```
mysql> SHOW CREATE TABLE employees \G
*************************** 1. row
***************************
       Table: employees
Create Table: CREATE TABLE `employees` (
  `emp_no` int(11) NOT NULL,
  `birth_date` date NOT NULL,
  `first_name` varchar(14) NOT NULL,
  `last_name` varchar(16) NOT NULL,
  `gender` enum('M','F') NOT NULL,
  `hire_date` date NOT NULL,
  `address` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`emp_no`,`hire_date`),
  KEY `name` (`first_name`,`last_name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
/*!50100 PARTITION BY RANGE (YEAR(hire_date))
(PARTITION p1980 VALUES LESS THAN (1980) ENGINE = InnoDB,
 PARTITION p1990 VALUES LESS THAN (1990) ENGINE = InnoDB,
 PARTITION p2000 VALUES LESS THAN (2000) ENGINE = InnoDB,
 PARTITION p2010 VALUES LESS THAN (2010) ENGINE = InnoDB,
 PARTITION p2020 VALUES LESS THAN (2020) ENGINE = InnoDB,
 PARTITION pmax VALUES LESS THAN MAXVALUE ENGINE = InnoDB)
*/
```

Using SHOW TABLE STATUS

You can execute the `SHOW TABLE STATUS` command and check `Create_options` in the output:

```
mysql> SHOW TABLE STATUS LIKE 'employees'\G
*************************** 1. row
***************************
           Name: employees
         Engine: InnoDB
        Version: 10
     Row_format: Dynamic
           Rows: NULL
 Avg_row_length: NULL
    Data_length: NULL
Max_data_length: NULL
   Index_length: NULL
      Data_free: NULL
 Auto_increment: NULL
    Create_time: 2017-10-01 05:01:53
    Update_time: NULL
     Check_time: NULL
      Collation: utf8mb4_0900_ai_ci
       Checksum: NULL
 Create_options: partitioned
        Comment:
1 row in set (0.00 sec)
```

Using EXPLAIN

The `EXPLAIN` plan shows all the partitions scanned for a query. If you run the `EXPLAIN` plan for `SELECT * FROM <table>`, it lists all the partitions, for example:

```
mysql> EXPLAIN SELECT * FROM employees\G
*************************** 1. row
***************************
           id: 1
  select_type: SIMPLE
        table: employees
   partitions: p1980,p1990,p2000,p2010,p2020,pmax
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 292695
     filtered: 100.00
        Extra: NULL
1 row in set, 1 warning (0.00 sec)
```

Querying the INFORMATION_SCHEMA.PARTITIONS table

Compared to all the preceding methods, `INFORMATION_SCHEMA.PARTITIONS` gives more information about the partitions:

```
mysql> SHOW CREATE TABLE INFORMATION_SCHEMA.PARTITIONS\G
*************************** 1. row
***************************
       Table: PARTITIONS
Create Table: CREATE TEMPORARY TABLE `PARTITIONS` (
  `TABLE_CATALOG` varchar(512) NOT NULL DEFAULT '',
  `TABLE_SCHEMA` varchar(64) NOT NULL DEFAULT '',
  `TABLE_NAME` varchar(64) NOT NULL DEFAULT '',
  `PARTITION_NAME` varchar(64) DEFAULT NULL,
  `SUBPARTITION_NAME` varchar(64) DEFAULT NULL,
  `PARTITION_ORDINAL_POSITION` bigint(21) unsigned DEFAULT
NULL,
  `SUBPARTITION_ORDINAL_POSITION` bigint(21) unsigned
DEFAULT NULL,
  `PARTITION_METHOD` varchar(18) DEFAULT NULL,
  `SUBPARTITION_METHOD` varchar(12) DEFAULT NULL,
  `PARTITION_EXPRESSION` longtext,
  `SUBPARTITION_EXPRESSION` longtext,
  `PARTITION_DESCRIPTION` longtext,
  `TABLE_ROWS` bigint(21) unsigned NOT NULL DEFAULT '0',
  `AVG_ROW_LENGTH` bigint(21) unsigned NOT NULL DEFAULT '0',
  `DATA_LENGTH` bigint(21) unsigned NOT NULL DEFAULT '0',
  `MAX_DATA_LENGTH` bigint(21) unsigned DEFAULT NULL,
  `INDEX_LENGTH` bigint(21) unsigned NOT NULL DEFAULT '0',
  `DATA_FREE` bigint(21) unsigned NOT NULL DEFAULT '0',
  `CREATE_TIME` datetime DEFAULT NULL,
  `UPDATE_TIME` datetime DEFAULT NULL,
  `CHECK_TIME` datetime DEFAULT NULL,
  `CHECKSUM` bigint(21) unsigned DEFAULT NULL,
  `PARTITION_COMMENT` varchar(80) NOT NULL DEFAULT '',
  `NODEGROUP` varchar(12) NOT NULL DEFAULT '',
  `TABLESPACE_NAME` varchar(64) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)
```

To find out more details about a table partition, you can query the `INFORMATION_SCHEMA.PARTITIONS` table by specifying the database name through `TABLE_SCHEMA` and table name through `TABLE_NAME`, for example:

```
mysql> SELECT PARTITION_NAME FROM
INFORMATION_SCHEMA.PARTITIONS WHERE TABLE_SCHEMA='employees'
AND TABLE_NAME='employees';
+----------------+
| PARTITION_NAME |
+----------------+
| p1980          |
| p1990          |
| p2000          |
| p2010          |
| p2020          |
| pmax           |
+----------------+
6 rows in set (0.00 sec)
```

You can get details such as `PARTITION_METHOD`, `PARTITION_EXPRESSION`, `PARTITION_DESCRIPTION`, and `TABLE_ROWS` in that partition:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.PARTITIONS WHERE
TABLE_SCHEMA='employees' AND TABLE_NAME='employees' AND
PARTITION_NAME='p1990'\G
*************************** 1. row
***************************
                TABLE_CATALOG: def
                 TABLE_SCHEMA: employees
                   TABLE_NAME: employees
               PARTITION_NAME: p1990
            SUBPARTITION_NAME: NULL
   PARTITION_ORDINAL_POSITION: 2
SUBPARTITION_ORDINAL_POSITION: NULL
             PARTITION_METHOD: RANGE
```

```
         SUBPARTITION_METHOD: NULL
        PARTITION_EXPRESSION: YEAR(hire_date)
     SUBPARTITION_EXPRESSION: NULL
       PARTITION_DESCRIPTION: 1990
                  TABLE_ROWS: 157588
              AVG_ROW_LENGTH: 56
                 DATA_LENGTH: 8929280
             MAX_DATA_LENGTH: NULL
                INDEX_LENGTH: 8929280
                   DATA_FREE: 0
                 CREATE_TIME: NULL
                 UPDATE_TIME: NULL
                  CHECK_TIME: NULL
                    CHECKSUM: NULL
           PARTITION_COMMENT:
                   NODEGROUP: default
             TABLESPACE_NAME: NULL
1 row in set (0.00 sec)
```

For more details, refer to https://dev.mysql.com/doc/refman/8.0/en/partitions-table.html.

Efficiently managing time to live and soft delete rows

`RANGE COLUMNS` is highly useful in managing time to live and soft delete rows. Suppose you have a application which specifies the expiry time of row (row to be deleted after it crosses the expiry time) and the expiry is varying.

Suppose the application can do the following types of inserts:

- Insert persistent data
- Insert with expiry

If the expiry is constant i.e all the rows inserted will be deleted after certain time, we can go with RANGE partitioning. But if the expiry is varying i.e some rows will be deleted in a week, some in a month, some in a year and some have no expiry, it is not possible to create partitions. In that case, you can use the `RANGE COLUMNS` partitioning explained below.

How it works...

We introduce a column called `soft_delete` which will be set by trigger. The `soft_delete` column will be part of range column partitioning.

Partitioning will be like (`soft_delete`, expires). The `soft_delete` and expires together controls which partition a row should go. soft_delete column decides the retention of the row. If the expires is 0, the trigger sets the `soft_delete` value to 0 which puts the row in a `no_retention` partition and if the value of expires is out of partition bounds, trigger sets the `soft_delete` value to 1 and the row will be put into a `long_retention` partition. If the value of expires is with in the partition bounds, trigger sets the `soft_delete` value to `2`. Depending on the value of the expires, the row will be put in respective partition.

To summarize, `soft_delete` will be :

- `0`: If the value of expires is 0
- `1`: If the expires is more than 30 days away from the timestamp
- `2`: If the expires is less than or equal to 30 days away from the timestamp

We create

- 1 `no_retention` partition (`soft_delete = 0`)
- 1 `long_retention` partition (`soft_delete = 1`)
- 8 daily partitions (`soft_delete = 2`)

How to do it...

You can create a table like this:

```
mysql> CREATE TABLE `customer_data` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `msg` text,
  `timestamp` bigint(20) NOT NULL DEFAULT '0',
  `expires` bigint(20) NOT NULL DEFAULT '0',
  `soft_delete` tinyint(3) unsigned NOT NULL DEFAULT '1',
  PRIMARY KEY (`id`,`expires`,`soft_delete`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
/*!50500 PARTITION BY RANGE COLUMNS(soft_delete,expires)
(PARTITION no_retention VALUES LESS THAN (0,MAXVALUE) ENGINE
= InnoDB,
 PARTITION long_retention VALUES LESS THAN (1,MAXVALUE)
ENGINE = InnoDB,
 PARTITION pd20171017 VALUES LESS THAN (2,1508198400000)
ENGINE = InnoDB,
 PARTITION pd20171018 VALUES LESS THAN (2,1508284800000)
ENGINE = InnoDB,
 PARTITION pd20171019 VALUES LESS THAN (2,1508371200000)
ENGINE = InnoDB,
 PARTITION pd20171020 VALUES LESS THAN (2,1508457600000)
ENGINE = InnoDB,
 PARTITION pd20171021 VALUES LESS THAN (2,1508544000000)
ENGINE = InnoDB,
 PARTITION pd20171022 VALUES LESS THAN (2,1508630400000)
ENGINE = InnoDB,
 PARTITION pd20171023 VALUES LESS THAN (2,1508716800000)
ENGINE = InnoDB,
 PARTITION pd20171024 VALUES LESS THAN (3,1508803200000)
ENGINE = InnoDB,
 PARTITION pd20171025 VALUES LESS THAN (3,1508869800000)
ENGINE = InnoDB,
 PARTITION pd20171026 VALUES LESS THAN (3,1508956200000)
ENGINE = InnoDB) */;
```

There will be a buffer weekly partition which will be 42 days away and will be always empty so that we can split and 7+2 daily partitions with 2 buffer.

```
mysql> DROP TRIGGER IF EXISTS customer_data_insert;
DELIMITER $$
CREATE TRIGGER customer_data_insert
BEFORE INSERT
   ON customer_data FOR EACH ROW
BEGIN
    SET NEW.soft_delete = (IF((NEW.expires =
0),0,IF((ROUND((((((NEW.expires - NEW.timestamp) / 1000) /
60) / 60) / 24),0) <= 7),2,1)));
END;
$$
DELIMITER ;
```

```
mysql> DROP TRIGGER IF EXISTS customer_data_update;
DELIMITER $$
CREATE TRIGGER customer_data_update
BEFORE UPDATE
   ON customer_data FOR EACH ROW
BEGIN
    SET NEW.soft_delete = (IF((NEW.expires =
0),0,IF((ROUND((((((NEW.expires - NEW.timestamp) / 1000) /
60) / 60) / 24),0) <= 7),2,1)));
END;
$$
DELIMITER ;
```

- Suppose the client inserts a row with timestamp of 1508265000 (2017-10-17 18:30:00) and expiry value of 1508351400 (2017-10-18 18:30:00), the soft_delete will be 2 which makes it into partition pd20171019

```
mysql> INSERT INTO customer_data(id, msg, timestamp,
expires) VALUES(1,'test',1508265000000,1508351400000);
Query OK, 1 row affected (0.05 sec)

```

```
mysql> SELECT * FROM customer_data PARTITION (pd20171019);
+----+------+---------------+---------------+-------------+
| id | msg  | timestamp     | expires       | soft_delete |
+----+------+---------------+---------------+-------------+
|  1 | test | 1508265000000 | 1508351400000 |           2 |
+----+-----+-------------+-------------+------------+
1 row in set (0.00 sec)
```

- Suppose the client does not set expiry, expires column will be 0 which makes the `soft_delete` to `0` and it will go to `no_retention` partition.

```
mysql> INSERT INTO customer_data(id, msg, timestamp,
expires)  VALUES(2,'non_expiry_row',1508265000000,0);
Query OK, 1 row affected (0.07 sec)
```

```
mysql> SELECT * FROM customer_data PARTITION (no_retention);
+----+----------------+---------------+---------+----------
--+
| id | msg            | timestamp     | expires |
soft_delete |
+----+----------------+---------------+---------+----------
--+
|  2 | non_expiry_row | 1508265000000 |       0 |
0 |
+----+----------------+---------------+---------+----------
--+
1 row in set (0.00 sec)
```

- Suppose the client wants to have expiry (assume 2017-10-19 06:30:00), expiry column can be updated which moves the row from `no_retention` partition to the respective partition (this has some performance impact because the row has to be moved across partitions)

```
mysql> UPDATE customer_data SET expires=1508394600000 WHERE
id=2;
Query OK, 1 row affected (0.06 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

```
mysql> SELECT * FROM customer_data PARTITION (no_retention);
Empty set (0.00 sec)
```

```
mysql> SELECT * FROM customer_data PARTITION (pd20171020);
+----+----------------+---------------+---------------+-----
--------+
| id | msg            | timestamp     | expires       |
soft_delete |
+----+----------------+---------------+---------------+-----
--------+
|  2 | non_expiry_row | 1508265000000 | 1508394600000 |
2 |
+----+----------------+---------------+---------------+-----
--------+
1 row in set (0.00 sec)
```

- Suppose the client sets a expiry which is beyond our partitions, it will automatically go into `long_retention` partition.

```
mysql> INSERT INTO customer_data(id, msg, timestamp,
expires)
VALUES(3,'long_expiry',1507852800000,1608025600000);

mysql> SELECT * FROM customer_data PARTITION
(long_retention);
+----+-------------+---------------+---------------+--------
-----+
| id | msg         | timestamp     | expires       |
soft_delete |
+----+-------------+---------------+---------------+--------
-----+
|  3 | long_expiry | 1507852800000 | 1608025600000 |
1 |
+----+-------------+---------------+---------------+--------
-----+
1 row in set (0.00 sec)
```

Movement of rows across partitions is slow, if you update the `soft_delete`, the row will be moved from default partition to other partition which will be slow.

Extending the logic

We can extend the logic and increase the value of `soft_delete` to accommodate more types of partitions.

- `0`: If the value of expires is 0
- `3`: If the expires is less than or equal to 7 days away from the timestamp
- `2`: If the expires is less than or equal to 60 days away from the timestamp
- `1`: If the expires is more than 60 days away from the timestamp

The `soft_delete` column will be part of partitioning. We create

- Single `no_retention` partition if value of `soft_delete` is `0`
- Single `long_retention` partition if the value of `soft_delete 1`
- Weekly partitions if the value of `soft_delete 2`
- Daily partitions if the value of `soft_delete 3`

Example partitioned table structure

There will be a buffer weekly partition which will be 42 days away and will be always empty so that we can split and 7+2 daily partitions with 2 buffer.

```
mysql> DROP TRIGGER IF EXISTS customer_data_insert;
DELIMITER $$
CREATE TRIGGER customer_data_insert
BEFORE INSERT
   ON customer_data FOR EACH ROW
BEGIN
    SET NEW.soft_delete = (IF((NEW.expires =
0),0,IF((ROUND((((((NEW.expires - NEW.timestamp) / 1000) /
60) / 60) / 24),0) <= 7),3,IF((ROUND((((((NEW.expires -
NEW.timestamp) / 1000) / 60) / 60) / 24),0) <= 42),2,1))));
END;
$$
DELIMITER ;
```

```
mysql> DROP TRIGGER IF EXISTS customer_data_update;
DELIMITER $$
CREATE TRIGGER customer_data_update
BEFORE INSERT
```

```
   ON customer_data FOR EACH ROW
BEGIN
    SET NEW.soft_delete = (IF((NEW.expires =
0),0,IF((ROUND((((((NEW.expires - NEW.timestamp) / 1000) /
60) / 60) / 24),0) <= 7),3,IF((ROUND((((((NEW.expires -
NEW.timestamp) / 1000) / 60) / 60) / 24),0) <= 42),2,1))));
END;
$$
DELIMITER ;

mysql> CREATE TABLE `customer_data` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `msg` text,
  `timestamp` bigint(20) NOT NULL DEFAULT '0',
  `expires` bigint(20) NOT NULL DEFAULT '0',
  `soft_delete` tinyint(3) unsigned NOT NULL DEFAULT '1',
  PRIMARY KEY (`id`,`expires`,`soft_delete`)
) ENGINE=InnoDB AUTO_INCREMENT=609585360 DEFAULT
CHARSET=utf8
/*!50500 PARTITION BY RANGE
COLUMNS(`soft_delete`,`expires`)
(
 PARTITION no_retention VALUES LESS THAN (0,MAXVALUE) ENGINE
= InnoDB,
 PARTITION long_retention VALUES LESS THAN (1,MAXVALUE)
ENGINE = InnoDB,
 PARTITION pw20171022 VALUES LESS THAN (2,1508630400000)
ENGINE = InnoDB,
 PARTITION pw20171029 VALUES LESS THAN (2,1509235200000)
ENGINE = InnoDB,
 PARTITION pw20171105 VALUES LESS THAN (2,1509840000000)
ENGINE = InnoDB,
 PARTITION pw20171112 VALUES LESS THAN (2,1510444800000)
ENGINE = InnoDB,
 PARTITION pw20171119 VALUES LESS THAN (2,1511049600000)
ENGINE = InnoDB,
 PARTITION pw20171126 VALUES LESS THAN (2,1511654400000)
ENGINE = InnoDB,
 PARTITION pw20171203 VALUES LESS THAN (2,1512259200000)
ENGINE = InnoDB,
 -- buffer partition which will be 67 days away and will be
always empty so that we can split
 PARTITION pw20171210 VALUES LESS THAN (2,1512864000000)
ENGINE = InnoDB,
 PARTITION pd20171016 VALUES LESS THAN (3,1508112000000)
ENGINE = InnoDB,
 PARTITION pd20171017 VALUES LESS THAN (3,1508198400000)
ENGINE = InnoDB,
 PARTITION pd20171018 VALUES LESS THAN (3,1508284800000)
ENGINE = InnoDB,
 PARTITION pd20171019 VALUES LESS THAN (3,1508371200000)
ENGINE = InnoDB,
 PARTITION pd20171020 VALUES LESS THAN (3,1508457600000)
ENGINE = InnoDB,
 PARTITION pd20171021 VALUES LESS THAN (3,1508544000000)
ENGINE = InnoDB,
 PARTITION pd20171022 VALUES LESS THAN (3,1508630400000)
ENGINE = InnoDB,
 PARTITION pd20171023 VALUES LESS THAN (3,1508716800000)
ENGINE = InnoDB,
 PARTITION pd20171024 VALUES LESS THAN (3,1508803200000)
ENGINE = InnoDB
 ) */;
```

Managing partitions

You can create a `CRON` in Linux or `EVENT` in mysql to manage the partitions. As the retention approaches, the partition management tool should reorganize the buffer partition into one usable partition and a buffer partition and also drop the partitions which crossed the retention.

For example, take the `customer_data` table mentioned previously.

On 20171203, you have to split the partition pw20171210 into pw20171210 and pw20171217.

On 20171017, you have to split the pd20171024 into pd20171024 and pd20171025.

Splitting(reorganizing) partitions would be very quick (~milli seconds if there are no queries locking the table) only if there is no (or very less) data. So we should aim to keep the partition empty by reorganizing it before the data enters the partition.

Managing Tablespace

In this chapter, we will cover the following recipes:

- Changing the number or size of InnoDB REDO log files
- Resizing the InnoDB system tablespace
- Creating file-per-table tablespaces outside the data directory
- Copying file-per-table tablespaces to another instance
- Managing UNDO tablespace
- Managing general tablespace
- Compressing InnoDB tables

Introduction

Before you begin this chapter, you should understand the basics of `InnoDB`.

As per the MySQL documentation,

System Tablespace (Shared tablespace)
""The InnoDB system tablespace contains the InnoDB data dictionary (metadata for InnoDB-related objects) and is the storage area for the doublewrite buffer, the change buffer, and undo logs. The system tablespace also contains table and index data for any user-created tables that are created in the system tablespace. The system tablespace is considered a shared tablespace since it is shared by multiple tables."

"The system tablespace is represented by one or more data files. By default, one system data file, named ibdata1, is created in the MySQL data directory. The size and number of system data files is controlled by the innodb_data_file_path startup option.""

File-per-table tablespace

A file-per-table tablespace is a single-table tablespace that is created in its own data file rather than in the system tablespace. Tables are created in file-per-table tablespaces when the `innodb_file_per_table` option is enabled. Otherwise, `InnoDB` tables are created in the system tablespace. Each file-per-table tablespace is represented by a single `.ibd` data file, which is created in the database directory by default.

File-per-table tablespaces support `DYNAMIC` and `COMPRESSED` row formats, which support features such as off-page storage for variable length data and table compression.

To know the advantages and disadvantages of file-per-table tablespaces, refer to https://dev.mysql.com/doc/refman/8.0/en/innodb-multiple-tablespaces.html and https://dev.mysql.com/doc/refman/8.0/en/innodb-parameters.html#sysvar_innodb_file_per_table.

General tablespace

A general tablespace is a shared `InnoDB` tablespace created using `CREATE TABLESPACE` syntax. General tablespaces can be created outside of the MySQL `data directory` , are capable of holding multiple tables, and support tables of all row formats.

UNDO tablespace

An undo log is a collection of undo log records associated with a single transaction. An undo log record contains information about how to undo the latest change by a transaction to a clustered index record. If another transaction needs to see the original data (as part of a consistent read operation), the unmodified data is retrieved from the undo log records. Undo logs exist within undo log segments, which are contained within rollback segments. Rollback segments reside in the system tablespace, temporary tablespace, and undo tablespaces.

An `UNDO` tablespace comprises one or more files that contain undo logs. The number of undo tablespaces used by `InnoDB` is defined by the `innodb_undo_tablespaces` configuration option.

These logs are used to roll back transactions and also for multi-version concurrency control.

Data dictionary

The `data dictionary` is metadata that keeps track of database objects such as tables, indexes, and table columns. For the MySQL `data dictionary`, introduced in MySQL 8.0, metadata is physically located in `InnoDB` file-per-table tablespace files in the MySQL database directory. For the `InnoDB data dictionary`, metadata is physically located in the `InnoDB` system tablespace.

MySQL data dictionary

MySQL server incorporates a transactional `data dictionary` that stores information about database objects. In previous MySQL releases, dictionary data was stored in metadata files, nontransactional tables, and storage engine-specific `data dictionaries`.

In previous MySQL releases, dictionary data was partially stored in metadata files. Issues with file-based metadata storage included expensive file scans, susceptibility to filesystem-related bugs, complex code for handling of replication and crash recovery failure states, and a lack of extensibility that made it difficult to add metadata for new features and relational objects.

Benefits of the MySQL `data dictionary` include:

- Simplicity of a centralized `data dictionary` schema that uniformly stores dictionary data
- Removal of file-based metadata storage
- Transactional, crash-safe storage of dictionary data

- Uniform and centralized caching for dictionary objects
- A simpler and improved implementation for some `INFORMATION_SCHEMA` tables
- Atomic DDL

The metadata files listed as follows are removed from MySQL. Unless otherwise noted, data previously stored in metadata files is now stored in `data dictionary` tables:

- `.frm` files: Table metadata files for table definition.
- `.par` files: Partition definition files. `InnoDB` stopped using the `.definition` partition files in MySQL 5.7 with the introduction of native partitioning support for `InnoDB` tables.
- `.trn` files: Trigger namespace files.
- `.trg` files: Trigger parameter files.
- `.isl` files: The `InnoDB` symbolic link files containing the location of file-per-table tablespace files created outside of the MySQL `data directory`.
- `db.opt` files: Database configuration files. These files, one per database directory, contain database default character set attributes.

The limitations of MySQL `data dictionary` are as follows:

- Manual creation of database directories under the `data directory` (for example, with `mkdir`) is unsupported. Manually created database directories are not recognized by the MySQL server.
- Moving data stored in MyISAM tables by copying and moving MyISAM data files is unsupported. Tables moved using this method are not discovered by the server.

- Simple backup and restore of individual MyISAM tables using copied data files is unsupported.
- DDL operations take longer due to writing to storage, undo logs, and redo logs instead of `.frm` files.

Transactional storage of dictionary data
The `data dictionary` schema stores dictionary data in transactional (`InnoDB`) tables. `data dictionary` tables are located in the `mysql` database together with `non-data dictionary` system tables.

`data dictionary` tables are created in a single `InnoDB` tablespace named `mysql.ibd` in the MySQL `data directory` . The `mysql.ibd` tablespace file must reside in the MySQL `data directory` and its name cannot be modified or used by another tablespace. Previously, these tables were created in individual tablespace files in the MySQL database directory.

Changing the number or size of InnoDB redo log files

The `ib_logfile0` file and `ib_logfile1` are the default `InnoDB` redo log files created inside the `data directory`, with 48 MB each. If you wish to change the size of the redo log files, you can simply change it in the configuration file and restart MySQL. In previous versions, you had to do a slow shutdown of MySQL server, remove the redo log files, change the config file, and then start MySQL server.

As of MySQL 8, `InnoDB` detects that the `innodb_log_file_size` differs from the redo log file size. It writes a log checkpoint, closes and removes the old log files, creates new log files at the requested size, and opens the new log files.

How to do it...

1. Check the sizes of the current files:

```
shell> sudo ls -lhtr /var/lib/mysql/ib_logfile*
-rw-r-----. 1 mysql mysql 48M Oct  7 10:16
/var/lib/mysql/ib_logfile1
-rw-r-----. 1 mysql mysql 48M Oct  7 10:18
/var/lib/mysql/ib_logfile0
```

2. Stop the MySQL server and make sure that it shuts down without errors:

```
shell> sudo systemctl stop mysqld
```

3. Edit the configuration file:

```
shell> sudo vi /etc/my.cnf
[mysqld]
innodb_log_file_size=128M
innodb_log_files_in_group=4
```

4. Start the MySQL server:

```
shell> sudo systemctl start mysqld
```

5. You can verify what MySQL did in the log file:

```
shell> sudo less /var/log/mysqld.log
2017-10-07T11:09:35.111926Z 1 [Warning] InnoDB:
Resizing redo log from 2*3072 to 4*8192 pages,
LSN=249633608
2017-10-07T11:09:35.213717Z 1 [Warning] InnoDB:
Starting to delete and rewrite log files.
2017-10-07T11:09:35.224724Z 1 [Note] InnoDB: Setting
log file ./ib_logfile101 size to 128 MB
2017-10-07T11:09:35.225531Z 1 [Note] InnoDB: Progress
```

```
in MB:
 100
2017-10-07T11:09:38.924955Z 1 [Note] InnoDB: Setting
log file ./ib_logfile1 size to 128 MB
2017-10-07T11:09:38.925173Z 1 [Note] InnoDB: Progress
in MB:
 100
2017-10-07T11:09:42.516065Z 1 [Note] InnoDB: Setting
log file ./ib_logfile2 size to 128 MB
2017-10-07T11:09:42.516309Z 1 [Note] InnoDB: Progress
in MB:
 100
2017-10-07T11:09:46.098023Z 1 [Note] InnoDB: Setting
log file ./ib_logfile3 size to 128 MB
2017-10-07T11:09:46.098246Z 1 [Note] InnoDB: Progress
in MB:
 100
2017-10-07T11:09:49.715400Z 1 [Note] InnoDB: Renaming
log file ./ib_logfile101 to ./ib_logfile0
2017-10-07T11:09:49.715497Z 1 [Warning] InnoDB: New log
files created, LSN=249633608
```

6. You can also see the new log files created:

```
shell> sudo ls -lhtr /var/lib/mysql/ib_logfile*
-rw-r-----. 1 mysql mysql 128M Oct  7 11:09
/var/lib/mysql/ib_logfile1
-rw-r-----. 1 mysql mysql 128M Oct  7 11:09
/var/lib/mysql/ib_logfile2
-rw-r-----. 1 mysql mysql 128M Oct  7 11:09
/var/lib/mysql/ib_logfile3
-rw-r-----. 1 mysql mysql 128M Oct  7 11:09
/var/lib/mysql/ib_logfile0
```

Resizing the InnoDB system tablespace

The `ibdata1` file in the `data directory` is the default system tablespace. You can configure `ibdata1` using the `innodb_data_file_path` and `innodb_data_home_dir` configuration options. The `innodb_data_file_path` configuration option is used to configure the `InnoDB` system tablespace data files. The value of `innodb_data_file_path` should be a list of one or more data file specifications. If you name two or more data files, separate them by semicolon (`;`) characters.

If you want a tablespace containing a fixed-size 50 MB data file named `ibdata1` and a 50 MB auto-extending file named `ibdata2` in the `data directory`, it can be configured like this:

```
shell> sudo vi /etc/my.cnf
[mysqld]
innodb_data_file_path=ibdata1:50M;ibdata2:50M:autoextend
```

If the `ibdata` files become so big, especially when `innodb_file_per_table` is not enabled and the disk becomes full, you might want to add another data file on another disk.

How to do it...

Resizing the `InnoDB` system tablespace is one topic that you would love to know more and more about. Let's get into its details.

Increasing the InnoDB system tablespace

Assuming that `innodb_data_file_path` is `ibdata1:50M:autoextend`, the size has reached 76 MB, and your disk is just 100 MB, you can add another disk and configure to add another tablespace onto the new disk:

1. Stop MySQL server:

```
shell> sudo systemctl stop mysql
```

2. Check the size of the existing `ibdata1` file:

```
shell> sudo ls -lhtr /var/lib/mysql/ibdata1
-rw-r----- 1 mysql mysql 76M Oct  6 13:33
/var/lib/mysql/ibdata1
```

3. Mount the new disk. Assuming it is mounted on `/var/lib/mysql_extend`, change the ownership to `mysql`; make sure that the file is not already created. If you are using AppArmour or SELinux, make sure you set the alias or context correctly:

```
shell> sudo chown mysql:mysql /var/lib/mysql_extend
shell> sudo chmod 750 /var/lib/mysql_extend
shell> sudo ls -lhtr /var/lib/mysql_extend
```

4. Open the `my.cnf` and add the following:

```
shell> sudo vi /etc/my.cnf
[mysqld]
innodb_data_home_dir=
innodb_data_file_path =
ibdata1:76M;/var/lib/mysql_extend/ibdata2:50M:autoexten
d
```

Since the existing size of `ibdata1` is 76 MB, you have to choose a maxvalue of at least 76 MB. The next `ibdata` file will be created on the new disk mounted on `/var/lib/mysql_extend/`. The `innodb_data_home_dir` option should be specified; otherwise, `mysqld` looks at a different path and fails with an error:

```
2017-10-07T06:30:00.658039Z 1 [ERROR] InnoDB: Operating
system error number 2 in a file operation.
2017-10-07T06:30:00.658084Z 1 [ERROR] InnoDB: The error
means the system cannot find the path specified.
2017-10-07T06:30:00.658088Z 1 [ERROR] InnoDB: If you
are installing InnoDB, remember that you must create
directories yourself, InnoDB does not create them.
2017-10-07T06:30:00.658092Z 1 [ERROR] InnoDB: File
.//var/lib/mysql_extend/ibdata2: 'create' returned OS
error 71. Cannot continue operation
```

5. Start MySQL server:

```
shell> sudo systemctl start mysql
```

6. Verify the new file. Since you have mentioned it as 50 MB, the initial size of the file would be 50 MB:

```
shell> sudo ls -lhtr /var/lib/mysql_extend/
total 50M
-rw-r-----. 1 mysql mysql 50M Oct  7 07:38 ibdata2
```

```
mysql> SHOW VARIABLES LIKE 'innodb_data_file_path';
+----------------------+-------------------------------
---------------------------+
| Variable_name        | Value
|
+----------------------+-------------------------------
---------------------------+
| innodb_data_file_path |
ibdata1:12M;/var/lib/mysql_extend/ibdata2:50M:autoexten
d |
+----------------------+-------------------------------
----------------------------+
1 row in set (0.00 sec)
```

Shrinking the InnoDB system tablespace

If you are not using `innodb_file_per_table`, then all of the table data is stored in system tablespace. If you drop a table, the space is not reclaimed. You can shrink the system tablespace and reclaim the disk space. This requires a major downtime, so it is recommended to perform the task on a slave by taking it out of rotation and then promoting it to master.

You can check the free space by querying the `INFORMATION_SCHEMA` tables:

```
mysql> SELECT SUM(data_free)/1024/1024 FROM
INFORMATION_SCHEMA.TABLES;
+--------------------------+
| sum(data_free)/1024/1024 |
+--------------------------+
|               6.00000000 |
+--------------------------+
1 row in set (0.00 sec)
```

1. Stop the writes to the database. If it is a master, `mysql> SET @@GLOBAL.READ_ONLY=1;`; if it is a slave, stop the replication and save the binary log coordinates:

```
     mysql> STOP SLAVE;
     mysql> SHOW SLAVE STATUS\G
```

2. Take a full backup using `mysqldump` or `mydumper`, excluding the `sys` database:

```
     shell> mydumper -u root --password=<password> --trx-
     consistency-only --kill-long-queries --long-query-guard
```

```
500 --regex '^(?!sys)' --outputdir /backups
```

3. Stop MySQL server:

```
shell> sudo systemctl stop mysql
```

4. Remove all the `*.ibd`, `*.ib_log`, and `ibdata` files. If you are using only `InnoDB` tables, you can wipe off the `data directory` and all the locations where system table spaces are stored (`innodb_data_file_path`):

```
shell> sudo rm -rf /var/lib/mysql/ib*
/var/lib/mysql/<database directories>
shell> sudo rm -rf /var/lib/mysql_extend/*
```

5. Initialize `data directory`:

```
shell> sudo mysqld --initialize --
datadir=/var/lib/mysql
shell> chown -R  mysql:mysql  /var/lib/mysql/
shell> chown -R  mysql:mysql  /var/lib/mysql_extend/
```

6. Get the temporary password:

```
shell> sudo grep "temporary password is generated"
/var/log/mysql/error.log | tail -1
2017-10-07T09:33:31.966223Z 4 [Note] A temporary
password is generated for root@localhost: lI-qerr5agpa
```

7. Start MySQL and change the password:

```
shell> sudo systemctl start mysqld
shell> mysql -u root -plI-qerr5agpa

mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY
'xxxx';
Query OK, 0 rows affected (0.01 sec)
```

8. Restore the backup. Use the temporary password to connect to MySQL:

```
shell> /opt/mydumper/myloader --directory=/backups/ --
queries-per-transaction=50000 --threads=6 --user=root -
-password=xxxx  --overwrite-tables
```

9. If it is a master, enable writes by `mysql> SET @@GLOBAL.READ_ONLY=0;`. If it is a slave, restore the replication by executing the `CHANGE MASTER TO COMMAND` and `START SLAVE;`.

Creating file-per-table tablespaces outside the data directory

In the previous section, you understood how to create a system tablespace in another disk. In this section, you will learn how to create an individual tablespace in another disk.

How to do it...

You can mount a new disk with particular performance or capacity characteristics, such as a fast SSD or a high-capacity HDD, onto a directory and configure `InnoDB` to use that. Within the destination directory, MySQL creates a subdirectory corresponding to the database name, and within that, a `.ibd` file for the new table. Remember, you cannot use the `DATA DIRECTORY` clause with the `ALTER TABLE` statement:

1. Mount the new disk and change the permissions. If you are using AppArmour or SELinux, make sure you set the alias or context correctly:

```
shell> sudo chown -R mysql:mysql
/var/lib/mysql_fast_storage
shell> sudo chmod 750 /var/lib/mysql_fast_storage
```

2. Create a table:

```
mysql> CREATE TABLE event_tracker (
event_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
event_name varchar(10),
ts timestamp NOT NULL,
event_type varchar(10)
)
TABLESPACE = innodb_file_per_table
DATA DIRECTORY = '/var/lib/mysql_fast_storage';
```

3. Check the `.ibd` file created in the new device:

```
shell> sudo ls -lhtr
/var/lib/mysql_fast_storage/employees/
total 128K
-rw-r-----. 1 mysql mysql 128K Oct  7 13:48
event_tracker.ibd
```

Copying file-per-table tablespaces to another instance

Copying the tablespace file (the `.ibd` file) is the fastest way of moving data around, rather than exporting and importing through `mysqldump` or `mydumper`. The data is available immediately rather than having to be reinserted and the indexes rebuilt. There are many reasons why you might copy an `InnoDB` file-per-table tablespace to a different instance:

- To run reports without putting extra load on a production server
- To set up identical data for a table on a new slave server
- To restore a backed-up version of a table or partition after a problem or mistake
- To have busy tables on an SSD device, or large tables on a high-capacity HDD device

How to do it...

The outline is: you create the table on the destination with the same table definition and execute the `DISCARD TABLESPACE` command on the destination. Execute `FLUSH TABLES FOR EXPORT` on the source, which ensures that changes to the named table have been flushed to disk, and so a binary table copy can be made while the instance is running. After that statement, the table is locked and does not accept any writes; however, reads can happen. You can copy the `.ibd` file of that table to the destination, execute `UNLOCK` tables on source, and finally execute the `IMPORT TABLESPACE` command, which accepts the copied `.ibd` file.

For example, you want to copy the `events_history` table in a test database from one server (source) to another server (destination).

Create `event_history` if not created already and insert a few rows for the demo:

```
mysql> USE test;
mysql> CREATE TABLE IF NOT EXISTS `event_history`(
  `event_id` int(11) NOT NULL,
  `event_name` varchar(10) DEFAULT NULL,
  `created_at` datetime NOT NULL,
  `last_updated` timestamp NULL DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP,
  `event_type` varchar(10) NOT NULL,
  `msg` tinytext NOT NULL,
  PRIMARY KEY (`event_id`,`created_at`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
PARTITION BY RANGE (to_days(`created_at`))
(PARTITION 2017_oct_week1 VALUES LESS THAN (736974) ENGINE =
InnoDB,
 PARTITION p20171008 VALUES LESS THAN (736975) ENGINE =
InnoDB,
 PARTITION p20171009 VALUES LESS THAN (736976) ENGINE =
InnoDB,
 PARTITION p20171010 VALUES LESS THAN (736977) ENGINE =
InnoDB,
 PARTITION p20171011 VALUES LESS THAN (736978) ENGINE =
InnoDB,
 PARTITION p20171012 VALUES LESS THAN (736979) ENGINE =
InnoDB,
 PARTITION p20171013 VALUES LESS THAN (736980) ENGINE =
InnoDB,
 PARTITION p20171014 VALUES LESS THAN (736981) ENGINE =
InnoDB,
 PARTITION p20171015 VALUES LESS THAN (736982) ENGINE =
InnoDB,
 PARTITION p20171016 VALUES LESS THAN (736983) ENGINE =
InnoDB,
 PARTITION p20171017 VALUES LESS THAN (736984) ENGINE =
InnoDB);
```

```
mysql> INSERT INTO event_history VALUES
(1,'test','2017-10-07','2017-10-08','click','test_message'),
(2,'test','2017-10-08','2017-10-08','click','test_message'),
(3,'test','2017-10-09','2017-10-09','click','test_message'),
(4,'test','2017-10-10','2017-10-10','click','test_message'),
(5,'test','2017-10-11','2017-10-11','click','test_message'),
(6,'test','2017-10-12','2017-10-12','click','test_message'),
(7,'test','2017-10-13','2017-10-13','click','test_message'),
(8,'test','2017-10-14','2017-10-14','click','test_message');
Query OK, 8 rows affected (0.01 sec)
Records: 8  Duplicates: 0  Warnings: 0
```

Copy full table

1. **On destination**: Create the table with the same definition as on the source:

```
mysql> USE test;
mysql> CREATE TABLE IF NOT EXISTS `event_history`(
  `event_id` int(11) NOT NULL,
  `event_name` varchar(10) DEFAULT NULL,
  `created_at` datetime NOT NULL,
  `last_updated` timestamp NULL DEFAULT
CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  `event_type` varchar(10) NOT NULL,
  `msg` tinytext NOT NULL,
  PRIMARY KEY (`event_id`,`created_at`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
PARTITION BY RANGE (to_days(`created_at`))
(PARTITION 2017_oct_week1 VALUES LESS THAN (736974)
ENGINE = InnoDB,
 PARTITION p20171008 VALUES LESS THAN (736975) ENGINE =
InnoDB,
 PARTITION p20171009 VALUES LESS THAN (736976) ENGINE =
InnoDB,
 PARTITION p20171010 VALUES LESS THAN (736977) ENGINE =
InnoDB,
 PARTITION p20171011 VALUES LESS THAN (736978) ENGINE =
InnoDB,
 PARTITION p20171012 VALUES LESS THAN (736979) ENGINE =
InnoDB,
 PARTITION p20171013 VALUES LESS THAN (736980) ENGINE =
InnoDB,
 PARTITION p20171014 VALUES LESS THAN (736981) ENGINE =
InnoDB,
 PARTITION p20171015 VALUES LESS THAN (736982) ENGINE =
InnoDB,
 PARTITION p20171016 VALUES LESS THAN (736983) ENGINE =
InnoDB,
 PARTITION p20171017 VALUES LESS THAN (736984) ENGINE =
InnoDB);
```

2. **On destination**: Discard the tablespace:

```
mysql> ALTER TABLE event_history DISCARD TABLESPACE;
Query OK, 0 rows affected (0.05 sec)
```

3. **On source**: Execute `FLUSH TABLES FOR EXPORT`:

```
mysql> FLUSH TABLES event_history FOR EXPORT;
Query OK, 0 rows affected (0.00 sec)
```

4. **On source**: Copy all the table-related files (`.ibd`, `.cfg`) from the `data directory` directory of the source to the `data directory` of the destination:

```
shell> sudo scp -i /home/mysql/.ssh/id_rsa
/var/lib/mysql/test/event_history#P#*
mysql@xx.xxx.xxx.xxx:/var/lib/mysql/test/
```

5. **On source**: Unlock the table for writes:

```
mysql> UNLOCK TABLES;
Query OK, 0 rows affected (0.00 sec)
```

6. **On destination**: Make sure that the ownership of the files is set to `mysql`:

```
shell> sudo ls -lhtr /var/lib/mysql/test
total 1.4M
-rw-r----- 1 mysql mysql 128K Oct  7 17:17
event_history#P#p20171017.ibd
-rw-r----- 1 mysql mysql 128K Oct  7 17:17
event_history#P#p20171016.ibd
-rw-r----- 1 mysql mysql 128K Oct  7 17:17
event_history#P#p20171015.ibd
-rw-r----- 1 mysql mysql 128K Oct  7 17:17
event_history#P#p20171014.ibd
-rw-r----- 1 mysql mysql 128K Oct  7 17:17
event_history#P#p20171013.ibd
-rw-r----- 1 mysql mysql 128K Oct  7 17:17
event_history#P#p20171012.ibd
```

```
-rw-r----- 1 mysql mysql 128K Oct  7 17:17
event_history#P#p20171011.ibd
-rw-r----- 1 mysql mysql 128K Oct  7 17:17
event_history#P#p20171010.ibd
-rw-r----- 1 mysql mysql 128K Oct  7 17:17
event_history#P#p20171009.ibd
-rw-r----- 1 mysql mysql 128K Oct  7 17:17
event_history#P#p20171008.ibd
-rw-r----- 1 mysql mysql 128K Oct  7 17:17
event_history#P#2017_oct_week1.ibd
```

7. **On destination**: Import the tablespace. You can ignore the warnings as long as the table definitions are the same. If you have copied the `.cfg` files as well, the warnings won't occur:

```
mysql> ALTER TABLE event_history IMPORT TABLESPACE;
Query OK, 0 rows affected, 12 warnings (0.31 sec)
```

8. **On destination**: Verify the data:

```
mysql> SELECT * FROM event_history;
+----------+------------+---------------------+--------
------------+-----------+-------------+
| event_id | event_name | created_at          |
last_updated        | event_type | msg          |
+----------+------------+---------------------+--------
------------+-----------+-------------+
|        1 | test       | 2017-10-07 00:00:00 | 2017-
10-08 00:00:00 | click      | test_message |
|        2 | test       | 2017-10-08 00:00:00 | 2017-
10-08 00:00:00 | click      | test_message |
|        3 | test       | 2017-10-09 00:00:00 | 2017-
10-09 00:00:00 | click      | test_message |
|        4 | test       | 2017-10-10 00:00:00 | 2017-
10-10 00:00:00 | click      | test_message |
|        5 | test       | 2017-10-11 00:00:00 | 2017-
10-11 00:00:00 | click      | test_message |
|        6 | test       | 2017-10-12 00:00:00 | 2017-
10-12 00:00:00 | click      | test_message |
|        7 | test       | 2017-10-13 00:00:00 | 2017-
10-13 00:00:00 | click      | test_message |
|        8 | test       | 2017-10-14 00:00:00 | 2017-
10-14 00:00:00 | click      | test_message |
+----------+-----------+--------------------+-------
------------+-----------+-------------+
8 rows in set (0.00 sec)
```

If you are doing it on a production system, to minimize downtime, you can copy to the files locally, which is very fast. Immediately execute `UNLOCK TABLES` and then copy the files to the destination. If you cannot afford downtime, you can use Percona XtraBackup, back up the single table, and apply the redo logs, which generate the `.ibd` files. You can copy them to the destination and import.

Copying individual partitions of a table

You added a new partition of the `events_history` table on the source and you wish to copy only the new partitions to the destination. For the sake of your understanding, create new partitions on the `events_history` table and insert a few rows:

```
mysql> ALTER TABLE event_history ADD PARTITION
(PARTITION p20171018 VALUES LESS THAN (736985) ENGINE =
InnoDB,
 PARTITION p20171019 VALUES LESS THAN (736986) ENGINE =
InnoDB);
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> INSERT INTO event_history VALUES
(9,'test','2017-10-17','2017-10-17','click','test_message'),
(10,'test','2017-10-18','2017-10-
18','click','test_message');
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM event_history PARTITION
(p20171018,p20171019);
+----------+------------+---------------------+------------
--------+------------+--------------+
| event_id | event_name | created_at          | last_updated
| event_type | msg          |
+----------+------------+---------------------+------------
--------+------------+--------------+
|        9 | test       | 2017-10-17 00:00:00 | 2017-10-17
00:00:00 | click      | test_message |
|       10 | test       | 2017-10-18 00:00:00 | 2017-10-18
00:00:00 | click      | test_message |
+----------+------------+---------------------+------------
--------+------------+--------------+
2 rows in set (0.00 sec)
```

Suppose you want to copy the newly created partition to the destination.

1. **On destination**: Create the partitions:

```
mysql> ALTER TABLE event_history ADD PARTITION
(PARTITION p20171018 VALUES LESS THAN (736985) ENGINE =
InnoDB,
 PARTITION p20171019 VALUES LESS THAN (736986) ENGINE =
InnoDB);
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

2. **On destination:** Discard only the partitions you want to import:

```
mysql> ALTER TABLE event_history DISCARD PARTITION
p20171018, p20171019 TABLESPACE;
 Query OK, 0 rows affected (0.06 sec)
```

3. **On source:** Execute `FLUSH TABLE FOR EXPORT`:

```
mysql> FLUSH TABLES event_history FOR EXPORT;
Query OK, 0 rows affected (0.01 sec)
```

4. **On source:** Copy the `.ibd` files of the partitions to the destination:

```
shell> sudo scp -i /home/mysql/.ssh/id_rsa \
/var/lib/mysql/test/event_history#P#p20171018.ibd \
/var/lib/mysql/test/event_history#P#p20171019.ibd \
mysql@35.198.210.229:/var/lib/mysql/test/
event_history#P#p20171018.ibd
100%  128KB 128.0KB/s   00:00
event_history#P#p20171019.ibd
100%  128KB 128.0KB/s   00:00
```

5. **On destination:** Make sure that the `.ibd` files of the required partitions are copied and have the owner as `mysql`:

```
shell> sudo ls -lhtr
/var/lib/mysql/test/event_history#P#p20171018.ibd
-rw-r----- 1 mysql mysql 128K Oct  7 17:54
/var/lib/mysql/test/event_history#P#p20171018.ibd

shell> sudo ls -lhtr
/var/lib/mysql/test/event_history#P#p20171019.ibd
-rw-r----- 1 mysql mysql 128K Oct  7 17:54
/var/lib/mysql/test/event_history#P#p20171019.ibd
```

6. **On destination:** Execute `IMPORT PARTITION TABLESPACE`:

```
mysql> ALTER TABLE event_history IMPORT PARTITION
p20171018, p20171019  TABLESPACE;
Query OK, 0 rows affected, 2 warnings (0.10 sec)
```

   You can ignore the warnings as long as the table definitions are the same. If you have copied the `.cfg` files also, the warning won't occur:

```
mysql> SHOW WARNINGS;
+---------+------+------------------------------------
------------------------------------------------------
------------------------------------------------------
-------------+
|  Level   | Code | Message
|
+---------+------+------------------------------------
------------------------------------------------------
------------------------------------------------------
-------------+
|  Warning | 1810 | InnoDB: IO Read error: (2, No such
file or directory) Error opening
'./test/event_history#P#p20171018.cfg', will attempt to
import without schema verification |
| Warning | 1810 | InnoDB: IO Read error: (2, No such
file or directory) Error opening
'./test/event_history#P#p20171019.cfg', will attempt to
import without schema verification |
+---------+------+------------------------------------
------------------------------------------------------
------------------------------------------------------
```

```
-------------+
2 rows in set (0.00 sec)
```

7. **On destination:** Verify the data:

```
mysql> SELECT * FROM event_history PARTITION
(p20171018,p20171019);
+----------+------------+---------------------+--------
------------+-----------+-------------+
| event_id | event_name | created_at          |
last_updated        | event_type | msg          |
+----------+------------+---------------------+--------
------------+-----------+-------------+
|        9 | test       | 2017-10-17 00:00:00 | 2017-
10-17 00:00:00 | click      | test_message |
|       10 | test       | 2017-10-18 00:00:00 | 2017-
10-18 00:00:00 | click      | test_message |
+----------+------------+---------------------+--------
------------+-----------+-------------+
2 rows in set (0.00 sec)
```

See also

Refer to https://dev.mysql.com/doc/refman/8.0/en/tablespace-copying.html to find out more about the limitations of this procedure.

Managing UNDO tablespace

You can manage the size of an `UNDO` tablespace through the dynamic variable `innodb_max_undo_log_size`, which is 1 GB by default, and the number of `UNDO` tablespaces through `innodb_undo_tablespaces`, which is 2 GB by default and dynamic from MySQL 8.0.2.

By default, `innodb_undo_log_truncate` is enabled. 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. For truncation to occur, there must be at least two undo tablespaces.

How to do it...

Verify the size of the UNDO logs:

```
shell> sudo ls -lhtr /var/lib/mysql/undo_00*
-rw-r-----. 1 mysql mysql 19M Oct  7 17:43
/var/lib/mysql/undo_002
-rw-r-----. 1 mysql mysql 16M Oct  7 17:43
/var/lib/mysql/undo_001
```

Suppose you want to reduce the files that are more than 15 MB. Remember, only one undo tablespace can be truncated. Selection of an undo tablespace for truncation is performed in a circular fashion to avoid truncating the same undo tablespace every time. After all the rollback segments in the undo tablespace are freed, the truncate operation runs and the undo tablespace is truncated to its initial size. The initial size of an undo tablespace file is 10 MB:

1. Make sure that the `innodb_undo_log_truncate` is enabled:

```
      mysql> SELECT @@GLOBAL.innodb_undo_log_truncate;
      +-----------------------------------+
      | @@GLOBAL.innodb_undo_log_truncate |
      +-----------------------------------+
      |                                 1 |
      +-----------------------------------+
      1 row in set (0.00 sec)
```

2. Set `innodb_max_undo_log_size` to 15 MB:

```
      mysql> SELECT @@GLOBAL.innodb_max_undo_log_size;
      +-----------------------------------+
      | @@GLOBAL.innodb_max_undo_log_size |
      +-----------------------------------+
      |                        1073741824 |
      +-----------------------------------+
      1 row in set (0.00 sec)

      mysql> SET
      @@GLOBAL.innodb_max_undo_log_size=15*1024*1024;
      Query OK, 0 rows affected (0.00 sec)

      mysql> SELECT @@GLOBAL.innodb_max_undo_log_size;
      +-----------------------------------+
      | @@GLOBAL.innodb_max_undo_log_size |
      +-----------------------------------+
      |                          15728640 |
      +-----------------------------------+
      1 row in set (0.00 sec)
```

3. An undo tablespace cannot be truncated until its rollback segments are freed. Normally, the purge system frees rollback segments once every 128 times that purge is invoked. To expedite the truncation of undo tablespaces, use the `innodb_purge_rseg_truncate_frequency` option to temporarily increase the frequency with which the purge system frees rollback segments:

```
mysql> SELECT
@@GLOBAL.innodb_purge_rseg_truncate_frequency;
+-----------------------------------------------+
| @@GLOBAL.innodb_purge_rseg_truncate_frequency |
+-----------------------------------------------+
|                                           128 |
+-----------------------------------------------+
1 row in set (0.00 sec)

mysql> SET
@@GLOBAL.innodb_purge_rseg_truncate_frequency=1;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT
@@GLOBAL.innodb_purge_rseg_truncate_frequency;
+-----------------------------------------------+
| @@GLOBAL.innodb_purge_rseg_truncate_frequency |
+-----------------------------------------------+
|                                             1 |
+-------------------------------------------------+
1 row in set (0.00 sec)
```

4. Usually on a busy system, at least one purge operation might have initiated and the truncation would have started. If you are practicing on your machine, you can initiate the purge by creating a big transaction:

```
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> DELETE FROM employees;
Query OK, 300025 rows affected (16.23 sec)

mysql> ROLLBACK;
Query OK, 0 rows affected (2.38 sec)
```

5. While the delete is in progress, you can watch the growth of the UNDO log files:

```
shell> sudo ls -lhtr /var/lib/mysql/undo_00*
-rw-r-----. 1 mysql mysql 19M Oct  7 17:43
/var/lib/mysql/undo_002
-rw-r-----. 1 mysql mysql 16M Oct  7 17:43
/var/lib/mysql/undo_001

shell> sudo ls -lhtr /var/lib/mysql/undo_00*
-rw-r-----. 1 mysql mysql 10M Oct  8 04:52
/var/lib/mysql/undo_001
-rw-r-----. 1 mysql mysql 27M Oct  8 04:52
/var/lib/mysql/undo_002

shell> sudo ls -lhtr /var/lib/mysql/undo_00*
-rw-r-----. 1 mysql mysql 10M Oct  8 04:52
/var/lib/mysql/undo_001
-rw-r-----. 1 mysql mysql 28M Oct  8 04:52
/var/lib/mysql/undo_002

shell> sudo ls -lhtr /var/lib/mysql/undo_00*
-rw-r-----. 1 mysql mysql 10M Oct  8 04:52
/var/lib/mysql/undo_001
-rw-r-----. 1 mysql mysql 29M Oct  8 04:52
```

```
/var/lib/mysql/undo_002

shell> sudo ls -lhtr /var/lib/mysql/undo_00*
-rw-r-----. 1 mysql mysql 10M Oct  8 04:52
/var/lib/mysql/undo_001
-rw-r-----. 1 mysql mysql 29M Oct  8 04:52
/var/lib/mysql/undo_002
```

You may notice that `undo_001` is truncated to 10 MB while `undo_002` is growing, accommodating the deleted rows of the `DELETE` statement.

6. After some time, you can notice that `unod_002` is also truncated to 10 MB:

```
shell> sudo ls -lhtr /var/lib/mysql/undo_00*
-rw-r-----. 1 mysql mysql 10M Oct  8 04:52
/var/lib/mysql/undo_001
-rw-r-----. 1 mysql mysql 10M Oct  8 04:54
/var/lib/mysql/undo_002
```

5. Once you've achieved the reduction of the `UNDO` tablespace, set the `innodb_purge_rseg_truncate_frequency` to a default of `128`:

```
mysql> SELECT
@@GLOBAL.innodb_purge_rseg_truncate_frequency;
+------------------------------------------------+
| @@GLOBAL.innodb_purge_rseg_truncate_frequency |
+------------------------------------------------+
|                                               1 |
+------------------------------------------------+
1 row in set (0.00 sec)

mysql> SET
@@GLOBAL.innodb_purge_rseg_truncate_frequency=128;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT
@@GLOBAL.innodb_purge_rseg_truncate_frequency;
+------------------------------------------------+
| @@GLOBAL.innodb_purge_rseg_truncate_frequency |
+-----------------------------------------------+
|                                           128 |
+-----------------------------------------------+
1 row in set (0.01 sec)
```

Managing general tablespace

Until MySQL 8, there were two types of tablespaces: system tablespace and individual tablespace. There are advantages and disadvantages of both types. To overcome the disadvantages, general tablespaces are introduced in MySQL 8. Similar to system tablespaces, general tablespaces are shared tablespaces that can store data for multiple tables. But you have fine control over a general tablespace. Multiple tables in fewer general tablespaces consume less memory for tablespace metadata than the same number of tables in separate file-per-table tablespaces.

The limitations are as follows:

- Similarly to the system tablespace, truncating or dropping tables stored in a general tablespace creates free space internally in the general tablespace `.ibd` data file, which can only be used for new `InnoDB` data. Space is not released back to the operating system as it is for file-per-table tablespaces.
- Transportable tablespaces are not supported for tables that belong to a general tablespace.

In this section, you will learn how to create a general tablespace and add and remove the tables from it.

Practical usage:
Initially, `InnoDB` maintains a `.frm` file, which contains table structure. MySQL needs to open and close the `.frm` file, which degrades performance. With MySQL 8, the `.frm` files are removed

and all of the metadata is handled using a transactional `data dictionary`. This enables the use of the general tablespace.

Suppose you are using MySQL 5.7 or earlier for SaaS or multi-tenant, where you have a separate schema for each customer and each customer has hundreds of tables. If your customers grow, you will notice the performance issues. But with the removal of `.frm` files from MySQL 8, the performance is greatly improved. Moreover, you can create a separate tablespace for each schema (customer).

How to do it...

Let's get started with creating it, first.

Create a general tablespace

You can create a general tablespace either in the MySQL `data directory` or outside it.

To create one in a MySQL `data directory`:

```
mysql> CREATE TABLESPACE `ts1` ADD DATAFILE 'ts1.ibd'
Engine=InnoDB;
Query OK, 0 rows affected (0.02 sec)
```

To create the tablespace outside it, mount the new disk on `/var/lib/mysql_general_ts` and change the ownership to `mysql`:

```
shell> sudo chown mysql:mysql /var/lib/mysql_general_ts

mysql> CREATE TABLESPACE `ts2` ADD DATAFILE
'/var/lib/mysql_general_ts/ts2.ibd' Engine=InnoDB;Query OK,
0 rows affected (0.02 sec)
```

Adding tables to a general tablespace

You can add a table to a tablespace while creating it, or you can run the `ALTER` command to move a table from one tablespace to another:

```
mysql> CREATE TABLE employees.table_gen_ts1 (id INT PRIMARY
KEY) TABLESPACE ts1;
Query OK, 0 rows affected (0.01 sec)
```

Suppose you want to move the `employees` table to `TABLESPACE ts2`:

```
mysql> USE employees;
Database changed

mysql> ALTER TABLE employees TABLESPACE ts2;
Query OK, 0 rows affected (3.93 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

You can notice the increase in the `ts2.ibd` file:

```
shell> sudo ls -lhtr /var/lib/mysql_general_ts/ts2.ibd
-rw-r-----. 1 mysql mysql 32M Oct  8 17:07
/var/lib/mysql_general_ts/ts2.ibd
```

Moving non-partitioned tables between tablespaces

You can move tables as follows:

1. This is how to move tables from one general tablespace to another.
   Suppose you want to move the `employees` table from `ts2` to `ts1`:

```
mysql> ALTER TABLE employees TABLESPACE ts1;
Query OK, 0 rows affected (3.83 sec)
Records: 0  Duplicates: 0  Warnings: 0

shell> sudo ls -lhtr /var/lib/mysql/ts1.ibd
-rw-r-----. 1 mysql mysql 32M Oct  8 17:16
/var/lib/mysql/ts1.ibd
```

2. This is how to move tables to file-per-table.
   Suppose you want to move the `employees` table from `ts1` to file per table:

```
mysql> ALTER TABLE employees TABLESPACE
innodb_file_per_table;
Query OK, 0 rows affected (4.05 sec)
Records: 0  Duplicates: 0  Warnings: 0

shell> sudo ls -lhtr
/var/lib/mysql/employees/employees.ibd
-rw-r-----. 1 mysql mysql 32M Oct  8 17:18
/var/lib/mysql/employees/employees.ibd
```

3. This is how to move tables to the system tablespace.
   Suppose you want to move the `employees` table from file per table to the system tablespace:

```
mysql> ALTER TABLE employees TABLESPACE innodb_system;
Query OK, 0 rows affected (5.28 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Managing partitioned tables in a general tablespace

You can create a table with partitions in multiple tablespaces:

```
mysql> CREATE TABLE table_gen_part_ts1 (id INT, value
varchar(100)) ENGINE = InnoDB
       PARTITION BY RANGE(id) (
        PARTITION p1 VALUES LESS THAN (1000000) TABLESPACE
ts1,
        PARTITION p2 VALUES LESS THAN (2000000) TABLESPACE
ts2,
        PARTITION p3 VALUES LESS THAN (3000000) TABLESPACE
innodb_file_per_table,
        PARTITION pmax VALUES LESS THAN (MAXVALUE)
TABLESPACE innodb_system);
Query OK, 0 rows affected (0.19 sec)
```

You can add a new partition in another tablespace, or if you do not mention anything, it will be created in table's default tablespace. An `ALTER TABLE tbl_name TABLESPACE tablespace_name` operation on a partitioned table only modifies the table's default tablespace. It does not move the table partitions. However, after changing the default tablespace, an operation that rebuilds the table (such as an `ALTER TABLE` operation that uses `ALGORITHM=COPY`) moves the partitions to the default tablespace if another tablespace is not defined explicitly using the `TABLESPACE` clause.

If you wish to move the partitions across a tablespace, you need to do `REORGANIZE` on the partition. For example, you want to move partition `p3` to `ts2`:

```
mysql> ALTER TABLE table_gen_part_ts1 REORGANIZE PARTITION
p3 INTO (PARTITION p3 VALUES LESS THAN (3000000) TABLESPACE
ts2);
```

Dropping general tablespace

You can use the `DROP TABLESPACE` command to drop the tablespace. However, all the tables inside that tablespace should be either dropped or moved:

```
mysql> DROP TABLESPACE ts2;
ERROR 3120 (HY000): Tablespace `ts2` is not empty.
```

You have to move the partitions `p2` and `p3` of table `table_gen_part_ts1` in the `ts2` tablespace to other tablespace before dropping:

```
mysql> ALTER TABLE table_gen_part_ts1 REORGANIZE PARTITION
p2 INTO (PARTITION p2 VALUES LESS THAN (3000000) TABLESPACE
ts1);

mysql> ALTER TABLE table_gen_part_ts1 REORGANIZE PARTITION
p3 INTO (PARTITION p3 VALUES LESS THAN (3000000) TABLESPACE
ts1);
```

Now you can drop the tablespace:

```
mysql> DROP TABLESPACE ts2;
Query OK, 0 rows affected (0.01 sec)
```

Compressing InnoDB tables

You can create tables where the data is stored in compressed form. Compression can help to improve both raw performance and scalability. Compression means less data is transferred between disk and memory, and it takes up less space on disk and in memory.

As per the MySQL documentation:

""Because processors and cache memories have increased in speed more than disk storage devices, many workloads are disk-bound. Data compression enables smaller database size, reduced I/O, and improved throughput, at the small cost of increased CPU utilization. Compression is especially valuable for read-intensive applications, on systems with enough RAM to keep frequently used data in memory. The benefits are amplified for tables with secondary indexes, because index data is compressed also.""

To enable compression, you need to create or alter the table with the `ROW_FORMAT=COMPRESSED KEY_BLOCK_SIZE` option. You can vary the `KEY_BLOCK_SIZE` parameter, which uses a smaller page size on disk than the configured `innodb_page_size` value. Compression won't work if the table is in system tablespace.

To create a compressed table in a general tablespace, `FILE_BLOCK_SIZE` must be defined for the general tablespace, which is specified when the tablespace is created. The `FILE_BLOCK_SIZE` value must be a valid compressed page size in relation to the `innodb_page_size` value, and the page size of the compressed table, defined by the `CREATE TABLE` or `ALTER TABLE KEY_BLOCK_SIZE` clause, must be equal to `FILE_BLOCK_SIZE/1024`.

In the buffer pool, the compressed data is held in small pages, with a page size based on the `KEY_BLOCK_SIZE` value. For extracting or updating column values, MySQL also creates an uncompressed page in the buffer pool with uncompressed data. Within the buffer pool, any updates to the uncompressed page are also rewritten back to the equivalent compressed page. You might need to size your buffer pool to accommodate the additional data of both compressed and uncompressed pages, although uncompressed pages are evicted from the buffer pool when space is needed and then uncompressed again on the next access.

When to use compression?

In general, compression works best on tables that include a reasonable number of character string columns and where data is read far more often than it is written. Because there are no guaranteed ways to predict whether or not compression benefits a particular situation, always test with a specific workload and dataset running on a representative configuration.

How to do it...

You need to choose the parameter `KEY_BLOCK_SIZE`. `innodb_page_size` is 16,000; ideally, half of that is 8,000, which is a good start. To tune compression, refer to https://dev.mysql.com/doc/refman/8.0/en/innodb-compression-tuning.html.

Enabling Compression for file_per_table Tables

1. Make sure that `file_per_table` is enabled:

```
mysql> SET GLOBAL innodb_file_per_table=1;
```

2. Specify the `ROW_FORMAT=COMPRESSED KEY_BLOCK_SIZE=8` in the create statement:

```
mysql> CREATE TABLE compressed_table (id INT PRIMARY
KEY) ROW_FORMAT=COMPRESSED KEY_BLOCK_SIZE=8;
Query OK, 0 rows affected (0.07 sec)
```

If the table already exists, you can execute `ALTER`:

```
mysql> ALTER TABLE event_history ROW_FORMAT=COMPRESSED
KEY_BLOCK_SIZE=8;
Query OK, 0 rows affected (0.67 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

If you try to compress a table that is in the system tablespace, you will get an error:

```
mysql> ALTER TABLE employees ROW_FORMAT=COMPRESSED
KEY_BLOCK_SIZE=8;
ERROR 1478 (HY000): InnoDB: Tablespace `innodb_system`
cannot contain a COMPRESSED table
```

Disabling Compression for file_per_table Tables

To disable compression, execute the `ALTER` table and specify `ROW_FORMAT=DYNAMIC` or `ROW_FORMAT=COMPACT`, followed by `KEY_BLOCK_SIZE=0`.

For example, if you do not want compression on the `event_history` table:

```
mysql> ALTER TABLE event_history ROW_FORMAT=DYNAMIC
KEY_BLOCK_SIZE=0;
Query OK, 0 rows affected (0.53 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Enabling Compression for General Tablespace

First, you need to create a compressed tablespace by mentioning `FILE_BLOCK_SIZE`; you cannot alter the tablespace's `FILE_BLOCK_SIZE`.

If you wish to create a compressed table, you need to create the table in the general tablespace, where the compression is enabled; moreover `KEY_BLOCK_SIZE` must be equal to `FILE_BLOCK_SIZE/1024`. If you do not mention `KEY_BLOCK_SIZE`, the value will be automatically taken from `FILE_BLOCK_SIZE`.

You can create multiple compressed general tablespaces with different `FILE_BLOCK_SIZE` values and just add the tables to the desired tablespace:

1. Create a general compressed tablespace. You can create one with `FILE_BLOCK_SIZE` of 8k and other with `FILE_BLOCK_SIZE` of 4k, and move all the tables with `KEY_BLOCK_SIZE` of 8 to 8k and 4 to 4k:

```
mysql> CREATE TABLESPACE `ts_8k` ADD DATAFILE
'ts_8k.ibd' FILE_BLOCK_SIZE = 8192 Engine=InnoDB;
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLESPACE `ts_4k` ADD DATAFILE
'ts_4k.ibd' FILE_BLOCK_SIZE = 4096 Engine=InnoDB;
Query OK, 0 rows affected (0.04 sec)
```

2. Create compressed tables in those tablespaces by mentioning the `ROW_FORMAT=COMPRESSED`:

```
mysql> CREATE TABLE compress_table_1_8k (id INT PRIMARY
KEY) TABLESPACE ts_8k ROW_FORMAT=COMPRESSED;
Query OK, 0 rows affected (0.01 sec)
```

If you do not mention `ROW_FORMAT=COMPRESSED`, you will get an error:

```
mysql> CREATE TABLE compress_table_2_8k (id INT PRIMARY
KEY) TABLESPACE ts_8k;
ERROR 1478 (HY000): InnoDB: Tablespace `ts_8k` uses
block size 8192 and cannot contain a table with
physical page size 16384
```

Optionally, you can mention the `KEY_BLOCK_SIZE=FILE_BLOCK_SIZE/1024`:

```
mysql> CREATE TABLE compress_table_8k (id INT PRIMARY
KEY) TABLESPACE ts_8k ROW_FORMAT=COMPRESSED
KEY_BLOCK_SIZE=8;
Query OK, 0 rows affected (0.01 sec)
```

If you mention anything other than `FILE_BLOCK_SIZE/1024`, you will get an error:

```
mysql> CREATE TABLE compress_table_2_8k (id INT PRIMARY
KEY) TABLESPACE ts_8k ROW_FORMAT=COMPRESSED
KEY_BLOCK_SIZE=4;
ERROR 1478 (HY000): InnoDB: Tablespace `ts_8k` uses
block size 8192 and cannot contain a table with
physical page size 4096
```

3. You can move the tables from the `file_per_table` tablespace to the compressed general tablespace only if the `KEY_BLOCK_SIZE` matches. Otherwise, you will get an error:

```
mysql> CREATE TABLE compress_tables_4k (id INT PRIMARY
KEY) TABLESPACE innodb_file_per_table
ROW_FORMAT=COMPRESSED KEY_BLOCK_SIZE=4;
Query OK, 0 rows affected (0.02 sec)
```

```
mysql> ALTER TABLE compress_tables_4k TABLESPACE ts_4k;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE compress_tables_4k TABLESPACE ts_8k;
ERROR 1478 (HY000): InnoDB: Tablespace `ts_8k` uses
block size 8192 and cannot contain a table with
physical page size 4096
```

Managing Logs

In this chapter, we will cover the following recipes:

- Managing the error log
- Managing the general query log and slow query log
- Managing the binary logs

Introduction

You will now learn about managing different type of logs: error log, general query log, slow query log, binary logs, relay logs, and DDL logs.

Managing the error log

As per the MySQL documentation:

The error log contains a record of `mysqld` startup and shutdown times. It also contains diagnostic messages such as errors, warnings, and notes that occur during server startup and shutdown, and while the server is running.

The error log subsystem consists of components that perform log event filtering and writing, as well as a system variable called `log_error_services` that configures which components to enable to achieve the desired logging result. The default value of `global.log_error_services` is `log_filter_internal; log_sink_internal`:

```
mysql> SELECT @@global.log_error_services;
+----------------------------------------+
| @@global.log_error_services            |
+----------------------------------------+
| log_filter_internal; log_sink_internal |
+----------------------------------------+
```

That value indicates that log events first pass through the built-in filter component, `log_filter_internal`, then through the built-in log writer component, `log_sink_internal`. Component order is significant because the server executes components in the order listed. Any loadable (not built in) component named in the `log_error_services` value must first be installed with `INSTALL COMPONENT` which will be described in this section.

To know about all types of error logging, refer to https://dev.mysql.com/doc/refman/8.0/en/error-log.html

How to do it...

Error logs are easy, in a way. Let's see how to configure an error log first.

Configuring the error log

The error logging is controlled by the `log_error` variable (`--log-error` for a startup script).

If `--log-error` is not given, the default destination is the console.
If `--log-error` is given without naming a file, the default destination is a file named `host_name.err` in the `data directory` .
If `--log-error` is given to name a file, the default destination is that file (with an `.err` suffix added if the name has no suffix), located under the `data directory` unless an absolute path name is given to specify a different location.

The `log_error_verbosity` system variable controls server verbosity for writing error, warning, and note messages to the error log. Permitted `log_error_verbosity` values are `1` (errors only), `2` (errors and warnings), and `3` (errors, warnings, and notes), with a default of `3`.

To change the error log location, edit the configuration file and restart MySQL:

```
shell> sudo mkdir /var/log/mysql
shell> sudo chown -R mysql:mysql /var/log/mysql

shell> sudo vi /etc/my.cnf
[mysqld]
log-error=/var/log/mysql/mysqld.log

shell> sudo systemctl restart mysql
```

Verify the error log:

```
mysql> SHOW VARIABLES LIKE 'log_error';
+---------------+--------------------------+
| Variable_name | Value                    |
+---------------+--------------------------+
| log_error     | /var/log/mysql/mysqld.log |
+---------------+--------------------------+
1 row in set (0.00 sec)
```

To adjust the verbosity, you can change the `log_error_verbosity` variable dynamically. However, it is recommended to keep the default value of `3` so that error, warning, and note messages are logged:

```
mysql> SET @@GLOBAL.log_error_verbosity=2;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @@GLOBAL.log_error_verbosity;
+------------------------------+
| @@GLOBAL.log_error_verbosity |
+------------------------------+
|                            2 |
+------------------------------+
1 row in set (0.00 sec)
```

Rotating the error log

Suppose the error log file has become bigger and you want to rotate it; you can simply move the file and execute the `FLUSH LOGS` command:

```
shell> sudo mv /var/log/mysql/mysqld.log
/var/log/mysql/mysqld.log.0;

shell> mysqladmin -u root -p<password> flush-logs
mysqladmin: [Warning] Using a password on the command line
interface can be insecure.

shell> ls -lhtr /var/log/mysql/mysqld.log
-rw-r-----. 1 mysql mysql 0 Oct 10 14:03
/var/log/mysql/mysqld.log

shell> ls -lhtr /var/log/mysql/mysqld.log.0
-rw-r-----. 1 mysql mysql 3.4K Oct 10 14:03
/var/log/mysql/mysqld.log.0
```

You can automate the preceding steps using some scripts and put them into `cron`.

If the location of an error log file is not writable by the server, the log-flushing operation fails to create a new log file:

```
shell> sudo mv /var/log/mysqld.log /var/log/mysqld.log.0 &&
mysqladmin flush-logs -u root -p<password>
mysqladmin: [Warning] Using a password on the command line
interface can be insecure.
mysqladmin: refresh failed; error: 'Unknown error'
```

Using the system log for logging

To use the system log for logging, you need to load the system log writer called `log_sink_syseventlog`. You can use the built-in filter, `log_filter_internal`, for filtering:

1. Load the system log writer:

```
mysql> INSTALL COMPONENT
'file://component_log_sink_syseventlog';
Query OK, 0 rows affected (0.43 sec)
```

2. Make it persistent across restarts:

```
mysql> SET PERSIST log_error_services =
'log_filter_internal; log_sink_syseventlog';
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'log_error_services';
+--------------------+------------------------------------
----------+
| Variable_name      | Value
|
+--------------------+------------------------------------
----------+
| log_error_services | log_filter_internal;
log_sink_syseventlog |
+--------------------+------------------------------------
----------+
1 row in set (0.00 sec)
```

3. You can verify that the logs will be directed to the syslog. On CentOS and Red Hat, you can check in `/var/log/messages`; on Ubuntu, you can check in `/var/log/syslog`.
   For the sake of a demo, the server was restarted. You can see those logs in the syslog:

```
      shell> sudo grep mysqld /var/log/messages | tail
      Oct 10 14:50:31 centos7 mysqld[20953]: InnoDB: Buffer
      pool(s) dump completed at 171010 14:50:31
      Oct 10 14:50:32 centos7 mysqld[20953]: InnoDB: Shutdown
      completed; log sequence number 350327631
      Oct 10 14:50:32 centos7 mysqld[20953]: InnoDB: Removed
      temporary tablespace data file: "ibtmp1"
      Oct 10 14:50:32 centos7 mysqld[20953]: Shutting down
      plugin 'MEMORY'
      Oct 10 14:50:32 centos7 mysqld[20953]: Shutting down
      plugin 'CSV'
      Oct 10 14:50:32 centos7 mysqld[20953]: Shutting down
      plugin 'sha256_password'
      Oct 10 14:50:32 centos7 mysqld[20953]: Shutting down
      plugin 'mysql_native_password'
      Oct 10 14:50:32 centos7 mysqld[20953]: Shutting down
      plugin 'binlog'
      Oct 10 14:50:32 centos7 mysqld[20953]:
      /usr/sbin/mysqld: Shutdown complete
      Oct 10 14:50:33 centos7 mysqld[21220]:
      /usr/sbin/mysqld: ready for connections. Version:
      '8.0.3-rc-log'  socket: '/var/lib/mysql/mysql.sock'
      port: 3306  MySQL Community Server (GPL)
```

If you have multiple `mysqld` processes running, you can differentiate using the PID specified in `[]`. Otherwise, you can set the `log_syslog_tag` variable, which appends the server identifier with a leading hyphen, resulting in an identifier of `mysqld-tag_val`. For example, you can tag an instance with something like `instance1`:

```
mysql> SELECT @@GLOBAL.log_syslog_tag;
+-------------------------+
| @@GLOBAL.log_syslog_tag |
+-------------------------+
|                         |
+-------------------------+
1 row in set (0.00 sec)

mysql> SET @@GLOBAL.log_syslog_tag='instance1';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT @@GLOBAL.log_syslog_tag;
+-------------------------+
| @@GLOBAL.log_syslog_tag |
+-------------------------+
| instance1               |
+-------------------------+
1 row in set (0.01 sec)

shell> sudo systemctl restart mysqld

shell> sudo grep mysqld /var/log/messages | tail
Oct 10 14:59:20 centos7 mysqld-instance1[21220]: InnoDB:
Buffer pool(s) dump completed at 171010 14:59:20
Oct 10 14:59:21 centos7 mysqld-instance1[21220]: InnoDB:
Shutdown completed; log sequence number 350355306
Oct 10 14:59:21 centos7 mysqld-instance1[21220]: InnoDB:
Removed temporary tablespace data file: "ibtmp1"
Oct 10 14:59:21 centos7 mysqld-instance1[21220]: Shutting
down plugin 'MEMORY'
Oct 10 14:59:21 centos7 mysqld-instance1[21220]: Shutting
down plugin 'CSV'
Oct 10 14:59:21 centos7 mysqld-instance1[21220]: Shutting
down plugin 'sha256_password'
Oct 10 14:59:21 centos7 mysqld-instance1[21220]: Shutting
down plugin 'mysql_native_password'
Oct 10 14:59:21 centos7 mysqld-instance1[21220]: Shutting
down plugin 'binlog'
Oct 10 14:59:21 centos7 mysqld-instance1[21220]:
/usr/sbin/mysqld: Shutdown complete
Oct 10 14:59:22 centos7 mysqld[21309]: /usr/sbin/mysqld:
ready for connections. Version: '8.0.3-rc-log'  socket:
'/var/lib/mysql/mysql.sock'  port: 3306  MySQL Community
Server (GPL)
```

You will notice that the `instance1` tag is appended to the log so that you can easily identify between multiple instances.

If you wish to switch back to the original logging, you can set `log_error_services` to `'log_filter_internal; log_sink_internal'`:

```
mysql> SET @@global.log_error_services='log_filter_internal;
log_sink_internal';
Query OK, 0 rows affected (0.00 sec)
```

Error logging in JSON format

To use the JSON format for logging, you need to load the JSON log writer called `log_sink_json`. You can use the built-in filter, `log_filter_internal`, for filtering:

1. Install the JSON log writer:

```
mysql> INSTALL COMPONENT
'file://component_log_sink_json';
Query OK, 0 rows affected (0.05 sec)
```

2. Make it persistent across restarts:

```
mysql> SET PERSIST log_error_services =
'log_filter_internal; log_sink_json';
Query OK, 0 rows affected (0.00 sec)
```

3. The JSON log writer determines its output destination based on the default error log destination, which is given by the `log_error` system variable:

```
mysql> SHOW VARIABLES LIKE 'log_error';
+---------------+--------------------------+
| Variable_name | Value                    |
+---------------+--------------------------+
| log_error     | /var/log/mysql/mysqld.log |
+---------------+--------------------------+
1 row in set (0.00 sec)
```

4. The log will be something like `mysqld.log.00.json`. After a restart, the JSON log file looks like this:

```
shell> sudo less /var/log/mysql/mysqld.log.00.json
{ "prio" : 2, "err_code" : 4356, "subsystem" : "",
"SQL_state" : "HY000", "source_file" : "sql_plugin.cc",
"function" : "reap_plugins", "msg" : "Shutting down
plugin 'sha256_password'", "time" : "2017-10-
15T12:29:08.862969Z", "err_symbol" :
"ER_PLUGIN_SHUTTING_DOWN_PLUGIN", "label" : "Note" }
{ "prio" : 2, "err_code" : 4356, "subsystem" : "",
"SQL_state" : "HY000", "source_file" : "sql_plugin.cc",
"function" : "reap_plugins", "msg" : "Shutting down
plugin 'mysql_native_password'", "time" : "2017-10-
15T12:29:08.862975Z", "err_symbol" :
"ER_PLUGIN_SHUTTING_DOWN_PLUGIN", "label" : "Note" }
{ "prio" : 2, "err_code" : 4356, "subsystem" : "",
"SQL_state" : "HY000", "source_file" : "sql_plugin.cc",
"function" : "reap_plugins", "msg" : "Shutting down
plugin 'binlog'", "time" : "2017-10-
15T12:29:08.863758Z",  "err_symbol" :
"ER_PLUGIN_SHUTTING_DOWN_PLUGIN", "label" : "Note" }
{  "prio" : 2, "err_code" : 1079, "subsystem" : "",
"SQL_state" : "HY000",  "source_file" : "mysqld.cc",
"function" : "clean_up", "msg" : "/usr/sbin/mysqld:
Shutdown complete\u000a", "time" : "2017-10-
15T12:29:08.867077Z", "err_symbol" :
"ER_SHUTDOWN_COMPLETE", "label" : "Note" }
{ "log_type" : 1, "prio" : 0, "err_code" : 1408, "msg"
: "/usr/sbin/mysqld: ready for connections. Version:
'8.0.3-rc-log'  socket: '/var/lib/mysql/mysql.sock'
port: 3306  MySQL Community Server (GPL)", "time" :
"2017-10-15T12:29:10.952502Z", "err_symbol" :
"ER_STARTUP", "SQL_state" : "HY000", "label" : "Note" }
```

If you wish to switch back to original logging, you can set `log_error_services` to `'log_filter_internal; log_sink_internal'`:

```
mysql> SET @@global.log_error_services='log_filter_internal;
log_sink_internal';
Query OK, 0 rows affected (0.00 sec)
```

To know more about the error logging configuration, refer to https://dev.mysql.com/doc/refman/8.0/en/error-log-component-configuration.html.

Managing the general query log and slow query log

There are two ways you can log the queries. One way is through general query log, and other way is through slow query log. In this section, you will learn about configuring them.

How to do it...

We will get into the details in the following subsections.

General query log

As per the MySQL documentation:

The general query log is a general record of what `mysqld` is doing. The server writes information to this log when clients connect or disconnect, and it logs each SQL statement received from clients. The general query log can be very useful when you suspect an error in a client and want to know exactly what the client sent to `mysqld`:

1. Specify the file for logging. If you do not specify, it will be created in the `data directory` with the name `hostname.log`. The server creates the file in the `data directory` unless an absolute path name is given to specify a different directory:

```
mysql> SET
@@GLOBAL.general_log_file='/var/log/mysql/general_query
_log';
Query OK, 0 rows affected (0.04 sec)
```

2. Enable the general query log:

```
mysql> SET GLOBAL general_log = 'ON';
Query OK, 0 rows affected (0.00 sec)
```

3. You can see that the queries are logged:

```
shell> sudo cat /var/log/mysql/general_query_log
/usr/sbin/mysqld, Version: 8.0.3-rc-log (MySQL
Community Server (GPL)). started with:
Tcp port: 3306  Unix socket: /var/lib/mysql/mysql.sock
Time                 Id Command    Argument
2017-10-11T04:21:00.118944Z      220 Connect
root@localhost on  using Socket
2017-10-11T04:21:00.119212Z      220 Query    select
```

```
@@version_comment limit 1
2017-10-11T04:21:03.217603Z      220 Query    SELECT
DATABASE()
2017-10-11T04:21:03.218275Z      220 Init DB
employees
2017-10-11T04:21:03.219339Z      220 Query    show
databases
2017-10-11T04:21:03.220189Z      220 Query    show
tables
2017-10-11T04:21:03.227635Z      220 Field List
current_dept_emp
2017-10-11T04:21:03.233820Z      220 Field List
departments
2017-10-11T04:21:03.235937Z      220 Field List
dept_emp
2017-10-11T04:21:03.236089Z      220 Field List
dept_emp_latest_date
2017-10-11T04:21:03.236337Z      220 Field List
dept_manager
2017-10-11T04:21:03.237291Z      220 Field List
employee_names
2017-10-11T04:21:03.237999Z      220 Field List
employees
2017-10-11T04:21:03.247921Z      220 Field List
titles
2017-10-11T04:21:03.248217Z      220 Field List
titles_only
~
~
2017-10-11T04:21:09.483117Z      220 Query    select
count(*) from employees
2017-10-11T04:21:10.523421Z      220 Quit
```

General query log generates a very big log file. Be very cautious when enabling it on a production server. It drastically affects the server's performance.

Slow query log

As per the MySQL documentation:

"The slow query log consists of SQL statements that took more than long_query_time seconds to execute and required at least min_examined_row_limit rows to be examined."

To log all the queries, you can set the value of `long_query_time` to `0`. The default value of `long_query_time` is `10` seconds and `min_examined_row_limit` is `0`.

By default, queries that do not use indexes for lookups and administrative statements (such as `ALTER TABLE`, `ANALYZE TABLE`, `CHECK TABLE`, `CREATE INDEX`, `DROP INDEX`, `OPTIMIZE TABLE`, and `REPAIR TABLE`) are not logged. This behavior can be changed using `log_slow_admin_statements` and `log_queries_not_using_indexes`.

To enable slow query log, you can dynamically set `slow_query_log=1` and you can set the filename using `slow_query_log_file`. To specify the log destination, use `--log-output`:

1. Verify `long_query_time` and adjust it as per your requirement:

```
mysql> SELECT @@GLOBAL.LONG_QUERY_TIME;
+--------------------------+
| @@GLOBAL.LONG_QUERY_TIME |
+--------------------------+
|                10.000000 |
+--------------------------+
1 row in set (0.00 sec)
```

```
mysql> SET @@GLOBAL.LONG_QUERY_TIME=1;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @@GLOBAL.LONG_QUERY_TIME;
+--------------------------+
| @@GLOBAL.LONG_QUERY_TIME |
+--------------------------+
|                 1.000000 |
+--------------------------+
1 row in set (0.00 sec)
```

2. Verify the slow query file. By default, it would be in the `data directory` with the `hostname-slow` log:

```
mysql> SELECT @@GLOBAL.slow_query_log_file;
+-----------------------------------+
| @@GLOBAL.slow_query_log_file      |
+-----------------------------------+
| /var/lib/mysql/server1-slow.log |
+-----------------------------------+
1 row in set (0.00 sec)

mysql> SET
@@GLOBAL.slow_query_log_file='/var/log/mysql/mysql_slow
.log';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @@GLOBAL.slow_query_log_file;
+---------------------------------+
| @@GLOBAL.slow_query_log_file  |
+---------------------------------+
| /var/log/mysql/mysql_slow.log |
+---------------------------------+
1 row in set (0.00 sec)

mysql> FLUSH LOGS;
Query OK, 0 rows affected (0.03 sec)
```

3. Enable the slow query log:

```
mysql> SELECT @@GLOBAL.slow_query_log;
+-------------------------+
| @@GLOBAL.slow_query_log |
+-------------------------+
|                       0 |
+-------------------------+
1 row in set (0.00 sec)

mysql> SET @@GLOBAL.slow_query_log=1;
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT @@GLOBAL.slow_query_log;
+-------------------------+
| @@GLOBAL.slow_query_log |
+-------------------------+
|                       1 |
+-------------------------+
1 row in set (0.00 sec)
```

4. Verify that the queries are logged (You have to execute few long running queries to see them in slow query log):

```
mysql> SELECT SLEEP(2);
+----------+
| SLEEP(2) |
+----------+
|        0 |
+----------+
1 row in set (2.00 sec)


shell> sudo less /var/log/mysql/mysql_slow.log
/usr/sbin/mysqld, Version: 8.0.3-rc-log (MySQL
Community Server (GPL)). started with:
Tcp port: 3306  Unix socket: /var/lib/mysql/mysql.sock
Time                 Id Command    Argument
# Time: 2017-10-15T12:43:55.038601Z
# User@Host: root[root] @ localhost []  Id:     7
# Query_time: 2.000845  Lock_time: 0.000000 Rows_sent:
1  Rows_examined: 0
SET timestamp=1508071435;
SELECT SLEEP(2);
```

Selecting query log output destinations

You can log the queries to either `FILE` or `TABLE` in MySQL itself by specifying the `log_output` variable, which can be either `FILE` or `TABLE`, or both `FILE` and `TABLE`.

If you specify `log_output` as `FILE`, the general query log and the slow query log will be written to the files specified by `general_log_file` and `slow_query_log_file`, respectively.

If you specify `log_output` as `TABLE`, the general query log and the slow query log will be written to the `mysql.general_log` and `mysql.slow_log` tables respectively. Log contents are accessible through SQL statements.

For example:

```
mysql> SET @@GLOBAL.log_output='TABLE';
Query OK, 0 rows affected (0.00 sec)

mysql> SET @@GLOBAL.general_log='ON';
Query OK, 0 rows affected (0.02 sec)
```

Execute a few queries and then query the `mysql.general_log` table:

```
mysql> SELECT * FROM mysql.general_log WHERE
command_type='Query' \G
~
~
*************************** 3. row
***************************
  event_time: 2017-10-25 10:56:56.416746
   user_host: root[root] @ localhost []
   thread_id: 2421
   server_id: 32
command_type: Query
    argument: show databases
*************************** 4. row
***************************
  event_time: 2017-10-25 10:56:56.418896
   user_host: root[root] @ localhost []
   thread_id: 2421
   server_id: 32
command_type: Query
    argument: show tables
*************************** 5. row
***************************
  event_time: 2017-10-25 10:57:08.207964
   user_host: root[root] @ localhost []
   thread_id: 2421
   server_id: 32
command_type: Query
    argument: select * from salaries limit 1
*************************** 6. row
***************************
  event_time: 2017-10-25 10:57:47.041475
   user_host: root[root] @ localhost []
   thread_id: 2421
   server_id: 32
command_type: Query
    argument: SELECT * FROM mysql.general_log WHERE
command_type='Query'
```

You can use the `slow_log` table in a similar way:

```
mysql> SET @@GLOBAL.slow_query_log=1;
Query OK, 0 rows affected (0.00 sec)

mysql> SET @@GLOBAL.long_query_time=1;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT SLEEP(2);
+----------+
| SLEEP(2) |
+----------+
|        0 |
+----------+
1 row in set (2.00 sec)

mysql> SELECT * FROM mysql.slow_log \G
*************************** 1. row
***************************
    start_time: 2017-10-25 11:01:44.817421
     user_host: root[root] @ localhost []
    query_time: 00:00:02.000530
     lock_time: 00:00:00.000000
     rows_sent: 1
 rows_examined: 0
            db: employees
last_insert_id: 0
     insert_id: 0
     server_id: 32
      sql_text: SELECT SLEEP(2)
     thread_id: 2421
1 row in set (0.00 sec)
```

If the slow query log table has become huge, you can rotate it by creating a new table and swapping it:

1. Create a new table, `mysql.general_log_new`:

```
      mysql> DROP TABLE IF EXISTS mysql.general_log_new;
      Query OK, 0 rows affected, 1 warning (0.19 sec)

      mysql> CREATE TABLE mysql.general_log_new LIKE
      mysql.general_log;
      Query OK, 0 rows affected (0.10 sec)
```

2. Swap the tables using the `RENAME TABLE` command:

```
      mysql> RENAME TABLE mysql.general_log TO
      mysql.general_log_1, mysql.general_log_new TO
      mysql.general_log;
      Query OK, 0 rows affected (0.00 sec)
```

Managing the binary logs

In this section, managing the binary logs in a replication environment is covered. Basic binary log handling is already covered in Chapter 6, *Binary Logging*, using the `PURGE BINARY LOGS` command and `expire_logs_days` variable.

Using those methods is unsafe in a replication environment because if any one of the slaves has not consumed the binary logs and you have deleted them, the slave will go out of sync and you'll need to rebuild it.

The safe way to delete the binary logs is by checking which binary logs have been read on each slave and deleting them. You can use the `mysqlbinlogpurge` utility to achieve this.

How to do it...

Execute the `mysqlbinlogpurge` script on any of the servers and specify the master and slave hosts. The script connects to all the slaves and finds out the latest binary log applied. Then it purges the master binary logs until that point. You need a superuser to connect to all slaves:

1. Connect to any of the servers and execute the `mysqlbinlogpurge` script:

```
shell> mysqlbinlogpurge --master=dbadmin:
<pass>@master:3306 --slaves=dbadmin:
<pass>@slave1:3306,dbadmin:<pass>@slave2:3306
```

```
mysql> SHOW BINARY LOGS;
+--------------------+-----------+
| Log_name           | File_size |
+--------------------+-----------+
| master-bin.000001  |       177 |
~
| master-bin.000018  |     47785 |
| master-bin.000019  |       203 |
| master-bin.000020  |       203 |
| master-bin.000021  |       177 |
| master-bin.000022  |       203 |
| master-bin.000023  |  57739432 |
+--------------------+-----------+
23 rows in set (0.00 sec)

shell> mysqlbinlogpurge --master=dbadmin:
<pass>@master:3306 --slaves=dbadmin:
<pass>@slave1:3306,dbadmin:<pass>@slave2:3306

# Latest binlog file replicated by all slaves: master-
bin.000022
# Purging binary logs prior to 'master-bin.000023'
```

2. If you wish to discover all the slaves without specifying in the command, you should set `report_host` and `report_port` on all slaves and restart MySQL server. On each slave:

```
shell> sudo vi /etc/my.cnf
[mysqld]
report-host     = slave1
report-port     = 3306

shell> sudo systemctl restart mysql

mysql> SHOW VARIABLES LIKE 'report%';
+-----------------+---------------+
| Variable_name   | Value         |
+-----------------+---------------+
| report_host     | slave1        |
| report_password |               |
| report_port     | 3306          |
| report_user     |               |
+-----------------+---------------+
4 rows in set (0.00 sec)
```

3. Execute `mysqlbinlogpurge` with the `discover-slaves-login` option:

```
mysql> SHOW BINARY LOGS;
+--------------------+-----------+
| Log_name           | File_size |
+--------------------+-----------+
| centos7-bin.000025 |       203 |
| centos7-bin.000026 |       203 |
| centos7-bin.000027 |       203 |
| centos7-bin.000028 |       154 |
+--------------------+-----------+
4 rows in set (0.00 sec)

shell> mysqlbinlogpurge --master=dbadmin:<pass>@master
--discover-slaves-login=dbadmin:<pass>
# Discovering slaves for master at master:3306
# Discovering slave at slave1:3306
# Found slave: slave1:3306
# Discovering slave at slave2:3306
# Found slave: slave2:3306
# Latest binlog file replicated by all slaves: master-
bin.000027
# Purging binary logs prior to 'master-bin.000028'
```

Performance Tuning

In this chapter, we will cover the following recipes:

- The explain plan
- Benchmarking queries and the server
- Adding indexes
- Invisible index
- Descending index
- Analyzing slow queries using pt-query-digest
- Optimizing datatypes
- Removing duplicate and redundant indexes
- Checking index usage
- Controlling the query optimizer
- Using index hints
- Indexing for JSON using generated columns
- Using resource groups
- Using performance_schema
- Using the sys schema

Introduction

This chapter will take you through query and schema tuning. The database is meant for the execution of queries; making it run faster is the end goal of tuning. The database's performance depends on many factors, mainly queries, schema, configuration settings, and hardware.

In this chapter, we are going to take the employees database to explain all the examples. You might have transformed the employees database in many ways in the preceding chapters. It is recommended to load the sample employees data again before trying the examples mentioned in this chapter. You can refer to *Section Loading sample data in Chapter 2* to know how to load sample data.

The explain plan

How MySQL executes queries is one of the major factors of database performance. You can verify the MySQL execution plan using the `EXPLAIN` command. Starting from MySQL 5.7.2, you can use `EXPLAIN` to examine queries currently executing in other sessions. `EXPLAIN FORMAT=JSON` gives detailed information.

How to do it...

Let's get into the details.

Using EXPLAIN

The explain plan gives information on how the optimizer is going to execute the query. You just need to prefix the `EXPLAIN` keyword to the query:

```
mysql> EXPLAIN SELECT dept_name FROM dept_emp JOIN employees
ON dept_emp.emp_no=employees.emp_no JOIN departments ON
departments.dept_no=dept_emp.dept_no WHERE
employees.first_name='Aamer'\G
*************************** 1. row
***************************
           id: 1
  select_type: SIMPLE
        table: employees
   partitions: NULL
         type: ref
possible_keys: PRIMARY,name
          key: name
      key_len: 58
          ref: const
         rows: 228
     filtered: 100.00
        Extra: Using index
*************************** 2. row
***************************
           id: 1
  select_type: SIMPLE
        table: dept_emp
   partitions: NULL
         type: ref
possible_keys: PRIMARY,dept_no
          key: PRIMARY
      key_len: 4
          ref: employees.employees.emp_no
         rows: 1
     filtered: 100.00
        Extra: Using index
*************************** 3. row
***************************
```

```
           id: 1
  select_type: SIMPLE
        table: departments
   partitions: NULL
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 16
          ref: employees.dept_emp.dept_no
         rows: 1
     filtered: 100.00
        Extra: NULL
3 rows in set, 1 warning (0.00 sec)
```

Using EXPLAIN JSON

Using the explain plan in JSON format gives complete information about the query execution:

```
mysql> EXPLAIN FORMAT=JSON SELECT dept_name FROM dept_emp
JOIN employees ON dept_emp.emp_no=employees.emp_no JOIN
departments ON departments.dept_no=dept_emp.dept_no WHERE
employees.first_name='Aamer'\G
*************************** 1. row
***************************
EXPLAIN: {
  "query_block": {
    "select_id": 1,
    "cost_info": {
      "query_cost": "286.13"
    },
    "nested_loop": [
      {
        "table": {
          "table_name": "employees",
          "access_type": "ref",
          "possible_keys": [
            "PRIMARY",
            "name"
          ],
          "key": "name",
          "used_key_parts": [
            "first_name"
          ],
          "key_length": "58",
          "ref": [
            "const"
          ],
          "rows_examined_per_scan": 228,
          "rows_produced_per_join": 228,
          "filtered": "100.00",
          "using_index": true,
          "cost_info": {
            "read_cost": "1.12",
            "eval_cost": "22.80",
            "prefix_cost": "23.92",
            "data_read_per_join": "30K"
          },
          "used_columns": [
            "emp_no",
            "first_name"
          ]
        }
      },
      {
        "table": {
          "table_name": "dept_emp",
          "access_type": "ref",
          "possible_keys": [
            "PRIMARY",
            "dept_no"
          ],
          "key": "PRIMARY",
          "used_key_parts": [
            "emp_no"
          ],
          "key_length": "4",
          "ref": [
            "employees.employees.emp_no"
          ],
          "rows_examined_per_scan": 1,
          "rows_produced_per_join": 252,
          "filtered": "100.00",
          "using_index": true,
          "cost_info": {
            "read_cost": "148.78",
            "eval_cost": "25.21",
            "prefix_cost": "197.91",
            "data_read_per_join": "7K"
          },
          "used_columns": [
            "emp_no",
            "dept_no"
          ]
        }
      },
      {
        "table": {
          "table_name": "departments",
          "access_type": "eq_ref",
          "possible_keys": [
            "PRIMARY"
          ],
          "key": "PRIMARY",
          "used_key_parts": [
            "dept_no"
          ],
          "key_length": "16",
          "ref": [
            "employees.dept_emp.dept_no"
          ],
          "rows_examined_per_scan": 1,
          "rows_produced_per_join": 252,
          "filtered": "100.00",
          "cost_info": {
            "read_cost": "63.02",
            "eval_cost": "25.21",
            "prefix_cost": "286.13",
            "data_read_per_join": "45K"
          },
          "used_columns": [
            "dept_no",
            "dept_name"
          ]
        }
      }
    ]
  }
}
```

Using EXPLAIN for connection

You can run the explain plan for an already-running session. You need to specify the connection ID:
To get connection ID, execute:

```
mysql> SELECT CONNECTION_ID();
+-----------------+
| CONNECTION_ID() |
+-----------------+
|             778 |
+-----------------+
1 row in set (0.00 sec)
```

```
mysql> EXPLAIN FORMAT=JSON FOR CONNECTION 778\G
*************************** 1. row
***************************
EXPLAIN: {
  "query_block": {
    "select_id": 1,
    "cost_info": {
      "query_cost": "881.04"
    },
    "nested_loop": [
      {
        "table": {
          "table_name": "employees",
          "access_type": "index",
          "possible_keys": [
            "PRIMARY"
          ],
          "key": "name",
          "used_key_parts": [
            "first_name",
            "last_name"
          ],
          "key_length": "124",
          "rows_examined_per_scan": 1,
          "rows_produced_per_join": 1,
          "filtered": "100.00",
          "using_index": true,
          "cost_info": {
            "read_cost": "880.24",
            "eval_cost": "0.10",
            "prefix_cost": "880.34",
            "data_read_per_join": "136"
          },
~
~
1 row in set (0.00 sec)
```

If the connection is not running any `SELECT`/`UPDATE`/`INSERT`/`DELETE`/`REPLACE` queries, it will throw an error:

```
mysql> EXPLAIN FOR CONNECTION 779;
ERROR 3012 (HY000): EXPLAIN FOR CONNECTION command is
supported only for SELECT/UPDATE/INSERT/DELETE/REPLACE
```

Refer to https://dev.mysql.com/doc/refman/8.0/en/explain-output.html to learn more about the explain plan format. The JSON format is very clearly explained at https://www.percona.com/blog/category/explain-2/explain-formatjson-is-cool/.

Benchmarking queries and the server

Suppose you want to find out which of the queries is faster. The explain plan gives you an idea, but sometimes you cannot decide based on it. You can execute them on the server and find which one is faster if the query time is in the order of tens of seconds. However, if the query time is in the order of a few milliseconds, you cannot decide based on a single execution.

You can use the `mysqlslap` utility (it comes along with MySQL-client installation), which emulates client load for a MySQL server and reports the timing of each stage. It works as if multiple clients are accessing the server. In this section, you will learn about the usage of `mysqlslap`; in later sections, you will learn about the power of the `mysqlslap`.

How to do it...

Suppose you want to measure the query time of a query; if you execute that in the MySQL client, you can know the approximate execution time with a granularity of 100 milliseconds:

```
mysql> pager grep rows
PAGER set to 'grep rows'
mysql> SELECT e.emp_no, salary FROM salaries s JOIN
employees e ON s.emp_no=e.emp_no WHERE (first_name='Adam');
2384 rows in set (0.00 sec)
```

You can emulate the client load using `mysqlslap` and run the preceding SQL concurrently over multiple iterations:

```
shell> mysqlslap -u <user> -p<pass> --create-
schema=employees --query="SELECT e.emp_no, salary FROM
salaries s JOIN employees e ON s.emp_no=e.emp_no WHERE
(first_name='Adam');" -c 1000 i 100
mysqlslap: [Warning] Using a password on the command line
interface can be insecure.
Benchmark
    Average number of seconds to run all queries: 3.216
seconds
    Minimum number of seconds to run all queries: 3.216
seconds
    Maximum number of seconds to run all queries: 3.216
seconds
    Number of clients running queries: 1000
    Average number of queries per client: 1
```

The preceding query was executed with 1,000 concurrencies and 100 iterations, and on average, it took 3.216 seconds.

You can specify multiple SQLs in a file and specify the delimiter. `mysqlslap` runs all the queries in the file:

```
shell> cat queries.sql
SELECT e.emp_no, salary FROM salaries s JOIN employees e ON
s.emp_no=e.emp_no WHERE (first_name='Adam');
SELECT * FROM employees WHERE first_name='Adam' OR
last_name='Adam';
SELECT * FROM employees WHERE first_name='Adam';

shell> mysqlslap -u <user> -p<pass> --create-
schema=employees --concurrency=10 --iterations=10 --
query=query.sql --query=queries.sql --delimiter=";"
mysqlslap: [Warning] Using a password on the command line
interface can be insecure.
Benchmark
    Average number of seconds to run all queries: 5.173
seconds
    Minimum number of seconds to run all queries: 5.010
seconds
    Maximum number of seconds to run all queries: 5.257
seconds
    Number of clients running queries: 10
    Average number of queries per client: 3
```

You can even autogenerate the table and SQL statements. In this way, you can compare the results with earlier server settings:

```
shell> mysqlslap -u <user> -p<pass> --concurrency=100 --
iterations=10 --number-int-cols=4 --number-char-cols=10  --
auto-generate-sql
mysqlslap: [Warning] Using a password on the command line
interface can be insecure.
Benchmark
    Average number of seconds to run all queries: 1.640
seconds
    Minimum number of seconds to run all queries: 1.511
seconds
    Maximum number of seconds to run all queries: 1.791
seconds
    Number of clients running queries: 100
    Average number of queries per client: 0
```

You can also use `performance_schema` for all query-related metrics, which is explained in the Using performance_schema section.

Adding indexes

Without an index, MySQL must scan the entire table row by row to find the relevant rows. If the table has an index on the columns that you are filtering for, MySQL can quickly find the rows in the big data file without scanning the whole file.

MySQL can use an index for filtering of rows in `WHERE`, `ORDER BY`, and `GROUP BY` clauses, and also for joining tables. If there are multiple indexes on a column, MySQL chooses the index that gives maximum filtering of rows.

You can execute the `ALTER TABLE` command to add or drop the index. Both index addition and dropping are online operations and do not hinder the DMLs on the table, but they take lot of time on larger tables.

Primary key (clustered index) and secondary indexes

Before you proceed further, it is important to understand what a primary key (or clustered index) is, and what a secondary index is.

`InnoDB` stores rows in a primary key in order to speed up queries and sorts involving the primary key columns. This is also called an **index-organized table**, in Oracle terms. All other indexes are referred to as secondary keys, which store the value of primary keys (they do not refer to the row directly).

Suppose the table is:

```
mysql> CREATE TABLE index_example (
col1 int PRIMARY KEY,
col2 char(10),
KEY `col2`(`col2`)
);
```

The table rows are sorted and stored based on the value of `col1`. If you search for any value of `col1`, it can directly point to the physical row; this is why a clustered index is lightning-fast. The index on `col2` also contains the value of `col1`, and if you search for `col2`, the value of `col1` is returned, which in turn is searched in the clustered index to return the actual row.

Tips on choosing the primary key:

- It should be `UNIQUE` and `NOT NULL`.
- Choose the smallest possible key because all the secondary indexes store the primary key. So if it is large, the overall

index size uses more space.

- Choose a monotonically increasing value. The physical rows are ordered based on the primary key. So if you choose a random key, more rearrangement of rows is needed, which leads to degraded performance. `AUTO_INCREMENT` is a perfect fit for primary key.
- Always choose a primary key; if you cannot find any, add an `AUTO_INCREMENT` column. If you do not choose any, `InnoDB` internally generates a hidden clustered index with a 6-byte row ID.

How to do it...

You can see the indexes of a table by viewing its definition. You will notice that there is an index on `first_name` and `last_name`. If you filter the rows by specifying `first_name` or by both (`first_name` and `last_name`), MySQL can use the index to speed up the query. However, if you specify only `last_name`, the index cannot be used; this is because the optimizer can only use any of the leftmost prefixes of the index. Refer to https://dev.mysql.com/doc/refman/8.0/en/multiple-column-indexes.html for more detailed examples:

```
mysql> ALTER TABLE employees ADD INDEX name(first_name,
last_name);
Query OK, 0 rows affected (2.23 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW CREATE TABLE employees\G
*************************** 1. row
***************************
       Table: employees
Create Table: CREATE TABLE `employees` (
  `emp_no` int(11) NOT NULL,
  `birth_date` date NOT NULL,
  `first_name` varchar(14) NOT NULL,
  `last_name` varchar(16) NOT NULL,
  `gender` enum('M','F') NOT NULL,
  `hire_date` date NOT NULL,
  PRIMARY KEY (`emp_no`),
  KEY `name` (`first_name`,`last_name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
1 row in set (0.00 sec)
```

Add index

You can add an index by executing the `ALTER TABLE ADD INDEX` command. For example, if you want to add an index on `last_name`, see the following code:

```
mysql> ALTER TABLE employees ADD INDEX (last_name);
Query OK, 0 rows affected (1.28 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW CREATE TABLE employees\G
*************************** 1. row
***************************
       Table: employees
Create Table: CREATE TABLE `employees` (
  `emp_no` int(11) NOT NULL,
  `birth_date` date NOT NULL,
  `first_name` varchar(14) NOT NULL,
  `last_name` varchar(16) NOT NULL,
  `gender` enum('M','F') NOT NULL,
  `hire_date` date NOT NULL,
  PRIMARY KEY (`emp_no`),
  KEY `name` (`first_name`,`last_name`),
  KEY `last_name` (`last_name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
1 row in set (0.01 sec)
```

You can specify the name of the index; if not, the leftmost prefix will be used as the name. If there are any duplicates, the name will be appended by `_2`, `_3`, and so on.

For example:

```
mysql> ALTER TABLE employees ADD INDEX index_last_name
(last_name);
```

UNIQUE index

If you want the index to be unique, you can specify the keyword `UNIQUE`. For example:

```
mysql> ALTER TABLE employees ADD UNIQUE INDEX unique_name
(last_name, first_name);
# There are few duplicate entries in employees database, the
above statement is shown for illustration purpose only.
```

Prefix index

For string columns, indexes that use only the leading part of column values, rather than the full column, can be created. You need to specify the length of the leading part:

```
## `last_name` varchar(16) NOT NULL
mysql> ALTER TABLE employees ADD INDEX (last_name(10));
Query OK, 0 rows affected (1.78 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

The maximum length of `last_name` is `16` characters, but the index is created only on the first 10 characters.

Drop index

You can drop an index using the `ALTER TABLE` command:

```
mysql> ALTER TABLE employees DROP INDEX last_name;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Index on generated columns

An index cannot be used on a column wrapped in a function. Suppose you add an index on `hire_date`:

```
mysql> ALTER TABLE employees ADD INDEX(hire_date);
Query OK, 0 rows affected (0.93 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

The index on `hire_date` can be used for queries having `hire_date` in the `WHERE` clause:

```
mysql> EXPLAIN SELECT COUNT(*) FROM employees WHERE
hire_date>'2000-01-01'\G
*************************** 1. row
***************************
           id: 1
  select_type: SIMPLE
        table: employees
   partitions: NULL
         type: range
possible_keys: hire_date
          key: hire_date
      key_len: 3
          ref: NULL
         rows: 14
     filtered: 100.00
        Extra: Using where; Using index
1 row in set, 1 warning (0.00 sec)
```

Instead, if you put `hire_date` inside a function, MySQL has to scan the full table:

```
mysql> EXPLAIN SELECT COUNT(*) FROM employees WHERE
YEAR(hire_date)>=2000\G
*************************** 1. row
***************************
           id: 1
```

```
  select_type: SIMPLE
        table: employees
   partitions: NULL
         type: index
possible_keys: NULL
          key: hire_date
      key_len: 3
          ref: NULL
         rows: 291892
     filtered: 100.00
        Extra: Using where; Using index
1 row in set, 1 warning (0.00 sec)
```

So try to avoid putting an indexed column inside a function. If you cannot avoid using a function, create a virtual column and add an index on the virtual column:

```
mysql> ALTER TABLE employees ADD hire_date_year YEAR AS
(YEAR(hire_date)) VIRTUAL, ADD INDEX (hire_date_year);
Query OK, 0 rows affected (1.16 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW CREATE TABLE employees\G
*************************** 1. row
***************************
       Table: employees
Create Table: CREATE TABLE `employees` (
  `emp_no` int(11) NOT NULL,
  `birth_date` date NOT NULL,
  `first_name` varchar(14) NOT NULL,
  `last_name` varchar(16) NOT NULL,
  `gender` enum('M','F') NOT NULL,
  `hire_date` date NOT NULL,
  `hire_date_year` year(4) GENERATED ALWAYS AS
(year(`hire_date`)) VIRTUAL,
  PRIMARY KEY (`emp_no`),
  KEY `name` (`first_name`,`last_name`),
  KEY `hire_date` (`hire_date`),
  KEY `hire_date_year` (`hire_date_year`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
1 row in set (0.00 sec)
```

Now, instead of using the `YEAR()` function in the query, you can directly use `hire_date_year` in the `WHERE` clause:

```
mysql> EXPLAIN SELECT COUNT(*) FROM employees WHERE
hire_date_year>=2000\G
*************************** 1. row
***************************
           id: 1
  select_type: SIMPLE
        table: employees
   partitions: NULL
         type: range
possible_keys: hire_date_year
          key: hire_date_year
      key_len: 2
          ref: NULL
         rows: 15
     filtered: 100.00
        Extra: Using where; Using index
1 row in set, 1 warning (0.00 sec)
```

Note that even if you use `YEAR(hire_date)`, the optimizer recognizes that the expression `YEAR()` matches the definition of `hire_date_year` and that `hire_date_year` is indexed; so it considers that index during execution plan construction:

```
mysql> EXPLAIN SELECT COUNT(*) FROM employees WHERE
YEAR(hire_date)>=2000\G
*************************** 1. row
***************************
           id: 1
  select_type: SIMPLE
        table: employees
   partitions: NULL
         type: range
possible_keys: hire_date_year
          key: hire_date_year
      key_len: 2
          ref: NULL
         rows: 15
     filtered: 100.00
```

```
         Extra: Using where
1 row in set, 1 warning (0.00 sec)
```

Invisible index

If you want to drop an unused index, then instead of dropping immediately, you can mark it as invisible, monitor the application behavior, and later drop it. Later, if you need that index, you can mark it as visible, which is very fast compared to dropping and re-adding indexes.

To explain the invisible index, you need to add normal index if not already there. Example:

```
mysql> ALTER TABLE employees ADD INDEX (last_name);
Query OK, 0 rows affected (1.81 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

How to do it...

If you wish to drop the index on `last_name`, rather than directly dropping, you can mark it as invisible using the `ALTER TABLE` command:

```
mysql> EXPLAIN SELECT * FROM employees WHERE
last_name='Aamodt'\G
*************************** 1. row
***************************
           id: 1
  select_type: SIMPLE
        table: employees
   partitions: NULL
         type: ref
possible_keys: last_name
          key: last_name
      key_len: 66
          ref: const
         rows: 205
     filtered: 100.00
        Extra: NULL
1 row in set, 1 warning (0.00 sec)

mysql> ALTER TABLE employees ALTER INDEX last_name
INVISIBLE;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN SELECT * FROM employees WHERE
last_name='Aamodt'\G
*************************** 1. row
***************************
           id: 1
  select_type: SIMPLE
        table: employees
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 299733
     filtered: 10.00
        Extra: Using where
1 row in set, 1 warning (0.00 sec)

mysql> SHOW CREATE TABLE employees\G
*************************** 1. row
***************************
       Table: employees
Create Table: CREATE TABLE `employees` (
  `emp_no` int(11) NOT NULL,
  `birth_date` date NOT NULL,
  `first_name` varchar(14) NOT NULL,
  `last_name` varchar(16) NOT NULL,
  `gender` enum('M','F') NOT NULL,
  `hire_date` date NOT NULL,
  PRIMARY KEY (`emp_no`),
  KEY `name` (`first_name`,`last_name`),
  KEY `last_name` (`last_name`) /*!80000 INVISIBLE */
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
1 row in set (0.00 sec)
```

You will notice that the query filtering through `last_name` is using the `last_name` index; after marking it as invisible, it is not able to use. You can mark it as visible again:

```
mysql> ALTER TABLE employees ALTER INDEX last_name VISIBLE;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Descending index

Prior to MySQL 8, an index definition could contain the order (ascending or descending), but it was only parsed and not implemented. The index values were always stored in ascending order. MySQL 8.0 introduced support for descending indexes. Thus, the specified order in the index definition is not ignored. A descending index actually stores key values in descending order. Remember that scanning an ascending index in reverse is not efficient for a descending query.

Consider a case where, in a multi-column index, you can specify certain columns to be descending. This can help for queries wherein we have both ascending and descending `ORDER BY` clauses.

Suppose you want to sort the `employees` table with `first_name` ascending and `last_name` descending; MySQL cannot use the index on `first_name` and `last_name`. Without a descending index:

```
mysql> SHOW CREATE TABLE employees\G
*************************** 1. row
***************************
       Table: employees
Create Table: CREATE TABLE `employees` (
  `emp_no` int(11) NOT NULL,
  `birth_date` date NOT NULL,
  `first_name` varchar(14) NOT NULL,
  `last_name` varchar(16) NOT NULL,
  `gender` enum('M','F') NOT NULL,
  `hire_date` date NOT NULL,
  PRIMARY KEY (`emp_no`),
  KEY `name` (`first_name`,`last_name`),
  KEY `last_name` (`last_name`) /*!80000 INVISIBLE */
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```

In the explain plan, you will notice that the index name (`first_name` and `last_name`) is not used:

```
mysql> EXPLAIN SELECT * FROM employees ORDER BY first_name
ASC, last_name DESC LIMIT 10\G
*************************** 1. row
***************************
           id: 1
  select_type: SIMPLE
        table: employees
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 299733
     filtered: 100.00
        Extra: Using filesort
```

How to do it...

1. Add a descending index:

```
mysql> ALTER TABLE employees ADD INDEX
name_desc(first_name ASC, last_name DESC);
Query OK, 0 rows affected (1.61 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

2. After adding the descending index, the query is able to use the index:

```
mysql> EXPLAIN SELECT * FROM employees ORDER BY
first_name ASC, last_name DESC LIMIT 10\G
*************************** 1. row
***************************
           id: 1
  select_type: SIMPLE
        table: employees
   partitions: NULL
         type: index
possible_keys: NULL
          key: name_desc
      key_len: 124
          ref: NULL
         rows: 10
     filtered: 100.00
        Extra: NULL
```

3. The same index can be used for the other way of ordering, that is, ordering by `first_name` descending and `last_name` ascending through the backward index scan:

```
mysql> EXPLAIN SELECT * FROM employees ORDER BY
first_name DESC, last_name ASC LIMIT 10\G
*************************** 1. row
***************************
           id: 1
```

```
  select_type: SIMPLE
        table: employees
   partitions: NULL
         type: index
possible_keys: NULL
          key: name_desc
      key_len: 124
          ref: NULL
         rows: 10
     filtered: 100.00
        Extra: Backward index scan
```

Analyzing slow queries using pt-query-digest

`pt-query-digest`, which is part of the Percona Toolkit, is used for analyzing queries. The queries can be collected through any of the following:

- Slow query log
- General query log
- Process list
- Binary log
- TCP dump

Installation of the Percona Toolkit is covered in Chapter 10, *Table Maintenance*, *Installing Percona Toolkit* section. In this section, you will learn how to use `pt-query-digest`. There are drawbacks for each method. The slow query log does not include all queries unless you specify `long_query_time` as `0`, which drastically slows down a system. The general query log does not include query time. You cannot get complete queries from the process list. Only writes can be analyzed using binary log and using TCP dump causes server degradation. Usually, this tool is used on the slow query log with a `long_query_time` of 1 second or higher.

How to do it...

Let's get into the details of analyzing slow queries using `pt-query-digest`.

Slow query log

Enabling and configuring the slow query log is explained in Chapter 12, *Managing Logs*, *Managing the general query log and slow query log*. Once the slow query log is enabled and queries are collected, you can run `pt-query-digest` by passing the slow query log.

Suppose the slow query file is at `/var/lib/mysql/mysql-slow.log`:

```
shell> sudo pt-query-digest /var/lib/mysql/ubuntu-slow.log >
query_digest
```

The digest report contains queries ranked by the number of query executions multiplied by the query time. Query details such as the query checksum (a unique value for each type of query), average time, percentage time, and number of executions are shown for all queries in the summary. You can drill down to the specific query by searching for the query checksum.

The digest report looks like the following:

```
# 286.8s user time, 850ms system time, 232.75M rss, 315.73M
vsz
# Current date: Sat Nov 18 05:16:55 2017
# Hostname: db1
# Files: /var/lib/mysql/db1-slow.log
# Rate limits apply
# Overall: 638.54k total, 2.06k unique, 0.49 QPS, 0.14x
concurrency ______
# Time range: 2017-11-03 01:02:40 to 2017-11-18 05:16:47
# Attribute          total     min     max     avg     95%
stddev  median
# ============     ======= ======= ======= ======= =======
======= =======
# Exec time        179486s     3us   2713s   281ms    21ms
15s   176us
# Lock time          1157s       0     36s     2ms   194us
124ms    49us
# Rows sent         18.25M       0 753.66k   29.96  212.52
1.63k    0.99
# Rows examine     157.39G       0   3.30G 258.45k   3.35k
24.78M    0.99
# Rows affecte       3.66M       0 294.77k    6.01    0.99
1.16k       0
# Bytes sent         3.08G       0  95.15M   5.05k  13.78k
206.42k  174.84
# Merge passes       2.84k       0      97    0.00       0
0.16       0
# Tmp tables       129.02k       0    1009    0.21    0.99
1.43       0
# Tmp disk tbl      25.20k       0     850    0.04       0
1.09       0
# Tmp tbl size      26.21G       0 218.27M  43.04k       0
2.06M       0
# Query size       178.92M       6 452.25k  293.81  592.07
5.26k   72.65
# InnoDB:
# IO r bytes        79.06G       0   2.09G 200.37k       0
12.94M       0
# IO r ops           7.26M       0 233.16k   18.39       0
1.36k       0
# IO r wait         96525s       0   3452s   233ms       0
18s       0
# pages distin     526.99M       0 608.33k   1.30k  964.41
9.15k    1.96
# queue wait             0       0       0       0       0
0       0
# rec lock wai         46s       0      9s   111us       0
28ms       0
# Boolean:
# Filesort       5% yes,  94% no
# Filesort on    0% yes,  99% no
# Full join      3% yes,  96% no
# Full scan     40% yes,  59% no
# Tmp table     13% yes,  86% no
# Tmp table on   2% yes,  97% no
```

The query profile will look like:

```
# Rank Query ID           Response time    Calls  R/Call
V/M   Item
# ==== ================== ================ ====== =========
===== ========
#    1 0x55F499860A034BCB 76560.4220 42.7%     47 1628.9451
18.06 SELECT orders
#    2 0x3A2F0B98DA39BCB9 10490.4155  5.8%   2680    3.9143
33... SELECT orders order_status
#    3 0x25119C7C31A24011  7378.8763  4.1%   1534    4.8102
30.11 SELECT orders users
#    4 0x41106CE92AD9DFED  5412.7326  3.0%  15589    0.3472
2.98 SELECT sessions
#    5 0x860DCDE7AE0AD554  5187.5257  2.9%    500   10.3751
54.99 SELECT orders sessions
#    6 0x5DF64920B008AD63  4517.5041  2.5%     58   77.8880
22.23 UPDATE SELECT
#    7 0xC9F9A31DE77B93A1  4473.0208  2.5%     58   77.1210
96... INSERT SELECT tmpMove
#    8 0x8BF88451DA989BFF  4036.4413  2.2%     13  310.4955
16... UPDATE SELECT orders tmpDel
```

From the preceding output, you can infer that for query `#1` (`0x55F499860A034BCB`), the cumulative response time for all executions is `76560` seconds. This accounts for 42.7% of the cumulative response time of all queries. The number of executions is 47 and the average query time is `1628` seconds.

You can go to any query by searching for the checksum. The complete query, commands for the explain plan, and the table status are displayed. For example:

```
# Query 1: 0.00 QPS, 0.06x concurrency, ID
0x55F499860A034BCB at byte 249542900
# This item is included in the report because it matches --
limit.
# Scores: V/M = 18.06
# Time range: 2017-11-03 01:39:19 to 2017-11-18 01:46:50
# Attribute    pct   total     min     max     avg     95%
stddev  median
# ============ === ======= ======= ======= ======= =======
======= =======
# Count           0      47
# Exec time      42  76560s   1182s   1854s   1629s   1819s
172s   1649s
# Lock time       0      3s   102us   994ms    70ms   293ms
174ms   467us
# Rows sent       0  78.78k     212   5.66k   1.68k   4.95k
1.71k  652.75
# Rows examine  85 135.34G   2.11G   3.30G   2.88G   3.17G
303.82M   2.87G
# Rows affecte   0       0       0       0       0       0
0       0
# Bytes sent      0   3.22M  10.20k 226.13k  70.14k 201.74k
66.71k  31.59k
# Merge passes   0       0       0       0       0       0
0       0
# Tmp tables      0       0       0       0       0       0
0       0
# Tmp disk tbl   0       0       0       0       0       0
0       0
# Tmp tbl size   0       0       0       0       0       0
0       0
# Query size      0  11.66k     254     254     254     254
0     254
# InnoDB:
# IO r bytes      1   1.11G       0  53.79M  24.20M  51.29M
21.04M  20.30M
# IO r ops        1 142.14k       0   6.72k   3.02k   6.63k
2.67k   2.50k
# IO r wait       0     92s       0     14s      2s      5s
3s      1s
# pages distin   0 325.46k   6.10k   7.30k   6.92k   6.96k
350.84   6.96k
# queue wait      0       0       0       0       0       0
0       0
# rec lock wai   0       0       0       0       0       0
0       0
# Boolean:
# Full scan    100% yes,   0% no
# String:
# Databases    lashrenew_... (32/68%), betsy_db (15/31%)
# Hosts        10.37.69.197
# InnoDB trxID CF22C985 (1/2%), CF23455A (1/2%)... 45 more
# Last errno   0
# rate limit   query:100
# Users        db1_... (32/68%), dba (15/31%)
# Query_time distribution
#   1us
#  10us
# 100us
#   1ms
#  10ms
# 100ms
#    1s
#  10s+
############################################################
####
# Tables
#    SHOW TABLE STATUS FROM `db1` LIKE 'orders'\G
#    SHOW CREATE TABLE `db1`.`orders`\G
#    SHOW TABLE STATUS FROM `db1` LIKE
'shipping_tracking_history'\G
#    SHOW CREATE TABLE `db1`.`shipping_tracking_history`\G
# EXPLAIN /*!50100 PARTITIONS*/
SELECT  tracking_num, carrier, order_id, userID FROM orders
o WHERE tracking_num!=""
and NOT EXISTS (SELECT 1 FROM shipping_tracking_history sth
WHERE sth.order_id=o.order_id AND sth.is_final=1)
AND o.date_finalized>date_add(curdate(),interval -1 month)\G
```

General query log

You can use `pt-query-digest` to analyze the general query log by passing the argument `--type genlog`. Since general logs do not report query times, only the count aggregate is shown:

```
 shell> sudo pt-query-digest --type genlog
/var/lib/mysql/db1.log   > general_query_digest
```

The output will be something like this:

```
# 400ms user time, 0 system time, 28.84M rss, 99.35M vsz
# Current date: Sat Nov 18 09:02:08 2017
# Hostname: db1
# Files: /var/lib/mysql/db1.log
# Overall: 511 total, 39 unique, 30.06 QPS, 0x concurrency
_______________
# Time range: 2017-11-18 09:01:09 to 09:01:26
# Attribute          total     min     max     avg     95%
stddev  median
# ============     ======= ======= ======= ======= =======
======= =======
# Exec time              0       0       0       0       0
0       0
# Query size        92.18k      10   3.22k  184.71  363.48
348.86  102.22
```

The query profile will look something like this:

```
# Profile
# Rank Query ID           Response time Calls R/Call V/M
Item
# ==== ================== ============= ===== ====== =====
===============
#    1 0x625BF8F82D174492  0.0000  0.0%   130 0.0000  0.00
SELECT facebook_like_details
#    2 0xAA353644DE4C4CB4  0.0000  0.0%    44 0.0000  0.00
ADMIN QUIT
```

```
#    3 0x5D51E5F01B88B79E  0.0000  0.0%    44 0.0000  0.00
ADMIN CONNECT
```

Process list

Instead of a log file, you can use `pt-query-digest` to read queries from the process list:

```
shell> pt-query-digest --processlist h=localhost  --
iterations 10 --run-time 1m -u <user> -p<pass>
```

`run-time` specifies how long each iteration should run. In the preceding example, the tool generates reports every minute for 10 minutes.

Binary log

To analyze the binary log using `pt-query-digest`, you should convert it to text format using the `mysqlbinlog` utility:

```
shell> sudo mysqlbinlog /var/lib/mysql/binlog.000639 >
binlog.00063

shell> pt-query-digest --type binlog binlog.000639  >
binlog_digest
```

TCP dump

You can capture TCP traffic using the `tcpdump` command and send it to `pt-query-digest` for analysis:

```
shell> sudo tcpdump -s 65535 -x -nn -q -tttt -i any -c 1000
port 3306 > mysql.tcp.txt

shell> pt-query-digest --type tcpdump mysql.tcp.txt >
tcpdump_digest
```

There are plenty of options available in `pt-query-digest`, such as filtering queries for a specific time window, filtering a specific query, and generating reports. Please refer to the Percona documentation at https://www.percona.com/doc/percona-toolkit/LATEST/pt-query-digest.html for more details.

See also

Refer to https://engineering.linkedin.com/blog/2017/09/query-analyzer--a-tool-for-analyzing-mysql-queries-without-overh to know more about the new way of analyzing all queries without any overhead.

Optimizing datatypes

You should define tables such that they occupy minimum space on disk while accommodating all possible values.

If the size is smaller:

- Less data is written to or read from the disk, which makes queries faster.
- The contents on the disk are loaded to the main memory while processing queries. So, smaller tables occupy less space in the main memory.
- Less space is occupied by indexes.

How to do it...

1. If you want to store an employee number, for which the maximum possible value is 500,000, the optimum datatype is `MEDIUMINT UNSIGNED` (which occupies 3 bytes). If you are storing it as `INT`, which occupies 4 bytes, you are wasting a byte for each row.
2. If you want to store the first name, for which the length is varying and the maximum possible value is 20, it is optimal to declare it as `varchar(20)`. If you are storing it as `char(20)`, and just a few names are 20 characters long while the remaining are less than 10 characters long, you are wasting space of 10 characters.
3. While declaring `varchar` columns, you should consider the length. Though `varchar` is optimized on-disk, while loading into memory, it occupies the full length. For example, if you store `first_name` in `varchar(255)` and the actual length is 10, on the disk it occupies 10 + 1 (an additional byte for storing length); but in the memory, it occupies the full length of 255 bytes.
4. If the length of the `varchar` column is more than 255 chars, it requires 2 bytes to store the length.
5. Declare the columns as `NOT NULL` if you are not storing null values. This avoids the overhead for testing whether each value is null and also saves some storage space: 1 bit per column.
6. If the length is fixed, use `char` instead of `varchar`, because `varchar` takes a byte or two to store the length of the string.
7. If the values are fixed, use `ENUM` rather than `varchar`. For example, if you want to store values that can be pending,

approved, rejected, deployed, undeployed, failed, or deleted, you can use `ENUM`. It takes 1 or 2 bytes, rather than `char(10)`, which occupies 10 bytes.
8. Prefer integers over strings.
9. Try to leverage the prefix index.
10. Try to leverage the `InnoDB` compression.

Refer to https://dev.mysql.com/doc/refman/8.0/en/storage-requirements.html to know more about the storage requirements of each datatype and https://dev.mysql.com/doc/refman/8.0/en/integer-types.html to know about the range of each integer type.

If you want to know the optimized datatype, you can use the function `PROCEDURE ANALYZE`. Though it is not accurate, it gives a fair idea of the fields. Unfortunately, it is deprecated in MySQL 8:

```
mysql> SELECT user_id, first_name FROM user PROCEDURE
ANALYSE(1,100)\G
*************************** 1. row
***************************
             Field_name: db1.user.user_id
              Min_value: 100000@nat.test123.net
              Max_value: test1234@nat.test123.net
             Min_length: 22
             Max_length: 33
       Empties_or_zeros: 0
                  Nulls: 0
Avg_value_or_avg_length: 25.8003
                    Std: NULL
      Optimal_fieldtype: VARCHAR(33) NOT NULL
*************************** 2. row
***************************
             Field_name: db1.user.first_name
              Min_value: *Alan
              Max_value: Zuniga 102031
             Min_length: 3
             Max_length: 33
       Empties_or_zeros: 0
                  Nulls: 0
```

```
Avg_value_or_avg_length: 10.1588
                    Std: NULL
      Optimal_fieldtype: VARCHAR(33) NOT NULL
2 rows in set (0.02 sec)
```

Removing duplicate and redundant indexes

You can define several indexes on a column. By mistake, you might have defined the same index again (same column, same order of columns, or same order of keys), which is called a **duplicate index**. If only partial indexes (leftmost columns) are repetitive, they are called **redundant indexes**. A duplicate index has no advantages. Redundant indexes might be useful in some cases (a use case is mentioned in the note at the end of this section), but both slow down the inserts. So, it is important to identify and remove them.

There are three tools that can help with finding out duplicate indexes:

- `pt-duplicate-key-checker`, which is part of the Percona Toolkit. Installing the Percona Toolkit is covered in Chapter 10, *Table Maintenance, Installing Percona Toolkit* section.
- `mysqlindexcheck`, which is part of MySQL utilities. Installing MySQL utilities is covered in Chapter 1, *MySQL 8.0 – Installing and Upgrading*.
- Using the `sys` schema, which will be covered in the next section.

Consider the following `employees` table:

```
mysql> SHOW CREATE TABLE employees\G
*************************** 1. row
***************************
       Table: employees
```

```
Create Table: CREATE TABLE `employees` (
  `emp_no` int(11) NOT NULL,
  `birth_date` date NOT NULL,
  `first_name` varchar(14) NOT NULL,
  `last_name` varchar(16) NOT NULL,
  `gender` enum('M','F') NOT NULL,
  `hire_date` date NOT NULL,
  PRIMARY KEY (`emp_no`),
  KEY `last_name` (`last_name`) /*!80000 INVISIBLE */,
  KEY `full_name` (`first_name`,`last_name`),
  KEY `full_name_desc` (`first_name` DESC,`last_name`),
  KEY `first_name` (`first_name`),
  KEY `full_name_1` (`first_name`,`last_name`),
  KEY `first_name_emp_no` (`first_name`,`emp_no`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```

The index `full_name_1` is the duplicate of `full_name` because both indexes are on the same columns, the same order of columns, and the same order of keys (ascending or descending).

The index `first_name` is a redundant index because the column `first_name` is already covered in the leftmost suffix of the `first_name` index.

The index `first_name_emp_no` is a redundant index because it contains the primary key in the rightmost suffix. The `InnoDB` secondary indexes already contain the primary key, so it is redundant to declare the primary key as part of the secondary index. However, it can be useful in queries that filter by `first_name` and sort by `emp_no`:

```
SELECT * FROM employees WHERE first_name='Adam' ORDER BY
emp_no;
```

The `full_name_desc` option is not a duplicate of `full_name` because the ordering of keys is different.

How to do it...

Let's get into the details of removing duplicate and redundant indexes.

pt-duplicate-key-checker

`pt-duplicate-key-checker` gives the exact `ALTER` statements to drop duplicate keys:

```
shell> pt-duplicate-key-checker -u <user> -p<pass>

# A software update is available:
#
########################################################################
# employees.employees
#
########################################################################

# full_name_1 is a duplicate of full_name
# Key definitions:
#   KEY `full_name_1` (`first_name`,`last_name`),
#   KEY `full_name` (`first_name`,`last_name`),
# Column types:
#      `first_name` varchar(14) not null
#      `last_name` varchar(16) not null
# To remove this duplicate index, execute:
ALTER TABLE `employees`.`employees` DROP INDEX `full_name_1`;

# first_name is a left-prefix of full_name
# Key definitions:
#   KEY `first_name` (`first_name`),
#   KEY `full_name` (`first_name`,`last_name`),
# Column types:
#      `first_name` varchar(14) not null
#      `last_name` varchar(16) not null
# To remove this duplicate index, execute:
ALTER TABLE `employees`.`employees` DROP INDEX `first_name`;

# Key first_name_emp_no ends with a prefix of the clustered index
# Key definitions:
#   KEY `first_name_emp_no` (`first_name`,`emp_no`)
#   PRIMARY KEY (`emp_no`),
# Column types:
#      `first_name` varchar(14) not null
#      `emp_no` int(11) not null
# To shorten this duplicate clustered index, execute:
ALTER TABLE `employees`.`employees` DROP INDEX `first_name_emp_no`, ADD
INDEX `first_name_emp_no` (`first_name`);
```

The tool suggests that you shorten the duplicate clustered index by removing the `PRIMARY KEY` from the rightmost suffix. Note that it may result in another duplicate index. If you wish to ignore the duplicate clustered indexes, you can pass the `--noclustered` option.

To check the duplicate indexes of a particular database, you can pass the `--databases <database name>` option:

```
shell> pt-duplicate-key-checker -u <user> -p<pass> --database employees
```

To drop the keys, you can even pipe the output of `pt-duplicate-key-checker` to `mysql`:

```
shell> pt-duplicate-key-checker -u <user> -p<pass> | mysql -u <user> -
p<pass>
```

mysqlindexcheck

Note that `mysqlindexcheck` ignores descending indexes. For example, `full_name_desc` (`first_name` descending and `last_name`) is treated as a duplicate index of `full_name` (`first_name` and `last_name`):

```
shell> mysqlindexcheck --server=<user>:<pass>@localhost:3306
employees --show-drops
WARNING: Using a password on the command line interface can
be insecure.
# Source on localhost: ... connected.
# The following indexes are duplicates or redundant for
table employees.employees:
#
CREATE INDEX `full_name_desc` ON `employees`.`employees`
(`first_name`, `last_name`) USING BTREE
#     may be redundant or duplicate of:
CREATE INDEX `full_name` ON `employees`.`employees`
(`first_name`, `last_name`) USING BTREE
#
CREATE INDEX `first_name` ON `employees`.`employees`
(`first_name`) USING BTREE
#     may be redundant or duplicate of:
CREATE INDEX `full_name` ON `employees`.`employees`
(`first_name`, `last_name`) USING BTREE
#
CREATE INDEX `full_name_1` ON `employees`.`employees`
(`first_name`, `last_name`) USING BTREE
#     may be redundant or duplicate of:
CREATE INDEX `full_name` ON `employees`.`employees`
(`first_name`, `last_name`) USING BTREE
#
# DROP statements:
#
ALTER TABLE `employees`.`employees` DROP INDEX
`full_name_desc`;
ALTER TABLE `employees`.`employees` DROP INDEX `first_name`;
ALTER TABLE `employees`.`employees` DROP INDEX
`full_name_1`;
```

```
#
# The following index for table employees.employees contains
the clustered index and might be redundant:
#
CREATE INDEX `first_name_emp_no` ON `employees`.`employees`
(`first_name`, `emp_no`) USING BTREE
#
# DROP/ADD statement:
#
ALTER TABLE `employees`.`employees` DROP INDEX
`first_name_emp_no`, ADD INDEX `first_name_emp_no`
(first_name);
#
```

As mentioned earlier, redundant indexes can be useful in some cases. You have to consider whether these kinds of cases are needed by your application.

Create indexes to understand the following examples

```
mysql> ALTER TABLE employees DROP PRIMARY KEY, ADD PRIMARY
KEY(emp_no, hire_date), ADD INDEX `name`
(`first_name`,`last_name`);

mysql> ALTER TABLE salaries ADD INDEX from_date(from_date),
ADD INDEX from_date_2(from_date,emp_no);
```

Consider the following `employees` and `salaries` tables:

```
mysql> SHOW CREATE TABLE employees\G
*************************** 1. row
***************************
       Table: employees
Create Table: CREATE TABLE `employees` (
  `emp_no` int(11) NOT NULL,
  `birth_date` date NOT NULL,
  `first_name` varchar(14) NOT NULL,
  `last_name` varchar(16) NOT NULL,
  `gender` enum('M','F') NOT NULL,
  `hire_date` date NOT NULL,
  PRIMARY KEY (`emp_no`,`hire_date`),
  KEY `name` (`first_name`,`last_name`)
) /*!50100 TABLESPACE `innodb_system` */ ENGINE=InnoDB
```

```
DEFAULT CHARSET=utf8mb4

mysql> SHOW CREATE TABLE salaries\G
*************************** 1. row
***************************
Create Table: CREATE TABLE `salaries` (
 `emp_no` int(11) NOT NULL,
 `salary` int(11) NOT NULL,
 `from_date` date NOT NULL,
 `to_date` date NOT NULL,
 PRIMARY KEY (`emp_no`,`from_date`),
 KEY `from_date` (`from_date`),
 KEY `from_date_2` (`from_date`,`emp_no`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```

It seems like `from_date` is a redundant index of `from_date_2`, but check the explain plan of the following query! It is using an intersection of both the indexes. The `from_date` index is used for filtering and `from_date_2` is used for joining with the `employees` table. The optimizer is scanning only one row in each table:

```
mysql> EXPLAIN SELECT e.emp_no, salary FROM salaries s JOIN
employees e ON s.emp_no=e.emp_no WHERE from_date='2001-05-
23'\G
*************************** 1. row
***************************
           id: 1
  select_type: SIMPLE
        table: s
   partitions: NULL
         type: index_merge
possible_keys: PRIMARY,from_date_2,from_date
          key: from_date_2,from_date
      key_len: 3,3
          ref: NULL
         rows: 1
     filtered: 100.00
        Extra: Using intersect(from_date_2,from_date); Using
where
*************************** 2. row
***************************
           id: 1
  select_type: SIMPLE
```

```
        table: e
   partitions: NULL
         type: ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: employees.s.emp_no
         rows: 1
     filtered: 100.00
        Extra: Using index
2 rows in set, 1 warning (0.00 sec)
```

Now drop the redundant index `from_date` and check the explain plan. You can see that the optimizer is scanning 90 rows in the `salaries` table and one row in the `employees` table. But look at the `ref` column; it shows that constants are compared to the index named in the `key` column (`from_date_2`) to select rows from the table. Rather than dropping the indexes, you can test this behavior by passing optimizer hints or index hints, which are covered in the next section:

```
mysql> EXPLAIN SELECT e.emp_no, salary FROM salaries s JOIN
employees e ON s.emp_no=e.emp_no WHERE from_date='2001-05-
23'\G
*************************** 1. row
***************************
           id: 1
  select_type: SIMPLE
        table: s
   partitions: NULL
         type: ref
possible_keys: PRIMARY,from_date_2
          key: from_date_2
      key_len: 3
          ref: const
         rows: 90
     filtered: 100.00
        Extra: NULL
*************************** 2. row
***************************
           id: 1
  select_type: SIMPLE
        table: e
   partitions: NULL
         type: ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: employees.s.emp_no
         rows: 1
     filtered: 100.00
        Extra: Using index
2 rows in set, 1 warning (0.00 sec)
```

Now you need to determine which of the queries is faster:

- **Plan 1**: Using `intersect(from_date, from_date_2)`; scanning one row with ref as null
- **Plan 2**: Using `from_date_2`; scanning 90 rows with ref as constant

You can use the `mysqlslap` utility to find that (do not run this directly on the production host) and make sure that the concurrency is less than `max_connections`.

The benchmark for plan 1 is as follows:

```
shell> mysqlslap -u <user> -p<pass> --create-
schema='employees' -c 500 -i 100 --query="SELECT e.emp_no,
salary FROM salaries s JOIN employees e ON s.emp_no=e.emp_no
WHERE from_date='2001-05-23'"
mysqlslap: [Warning] Using a password on the command line
interface can be insecure.
Benchmark
    Average number of seconds to run all queries: 0.466
seconds
    Minimum number of seconds to run all queries: 0.424
seconds
    Maximum number of seconds to run all queries: 0.568
seconds
```

```
    Number of clients running queries: 500
    Average number of queries per client: 1
```

The benchmark for plan 2 is:

```
shell> mysqlslap -u <user> -p<pass> --create-
schema='employees' -c 500 -i 100 --query="SELECT e.emp_no,
salary FROM salaries s JOIN employees e ON s.emp_no=e.emp_no
WHERE from_date='2001-05-23'"
mysqlslap: [Warning] Using a password on the command line
interface can be insecure.
Benchmark
    Average number of seconds to run all queries: 0.435
seconds
    Minimum number of seconds to run all queries: 0.376
seconds
    Maximum number of seconds to run all queries: 0.504
seconds
    Number of clients running queries: 500
    Average number of queries per client: 1
```

It turns out that the average query time for plan 1 and plan 2 are 0.466 seconds and 0.435 seconds, respectively. Since the results are very close, you can take a call and drop the redundant index. Use plan 2.

This is just an example that will enable you to learn and apply the concept in your application scenarios.

Checking index usage

In the preceding section, you learned about removing redundant and duplicate indexes. While designing an application, you might have thought about filtering a query based on a column and added index. But over a period of time, because of changes in the application, you might not need that index. In this section, you will learn about identifying those unused indexes.

There are two ways you can find unused indexes:

- Using `pt-index-usage` (covered in this section)
- Using `sys` schema (covered in the next section)

How to do it...

We can use the `pt-index-usage` tool from the Percona Toolkit to get the index analysis. It takes queries from the slow query log, runs the explain plan for each and every query, and identifies the unused indexes. If you have a list of queries, you can save them in slow query format and pass that to the tool. Note that this is only an approximation because the slow query log does not include all the queries:

```
shell> sudo pt-index-usage slow -u <user> -p<password>
/var/lib/mysql/db1-slow.log > unused_indexes
```

Controlling the query optimizer

The task of the query optimizer is to find an optimal plan for executing a SQL query. There can be multiple plans to execute a query, especially when joining a table, where the number of plans to be examined increases exponentially. In this section, you will learn about adjusting the optimizer to your needs.

Take the example of the `employees` table and add the necessary index;

```
mysql> CREATE TABLE `employees_index_example` (
   `emp_no` int(11) NOT NULL,
   `birth_date` date NOT NULL,
   `first_name` varchar(14) NOT NULL,
   `last_name` varchar(16) NOT NULL,
   `gender` enum('M','F') NOT NULL,
   `hire_date` date NOT NULL,
  PRIMARY KEY (`emp_no`),
  KEY `last_name` (`last_name`) /*!80000 INVISIBLE */,
  KEY `full_name` (`first_name`,`last_name`),
  KEY `full_name_desc` (`first_name` DESC,`last_name`),
  KEY `first_name` (`first_name`),
  KEY `full_name_1` (`first_name`,`last_name`),
  KEY `first_name_emp_no` (`first_name`,`emp_no`),
  KEY `last_name_2` (`last_name`(10))
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
Query OK, 0 rows affected, 1 warning (0.08 sec)

mysql> SHOW WARNINGS;
+---------+------+------------------------------------------
------------------------------------------------------------
----------------------------------------+
| Level   | Code | Message
|
+---------+------+------------------------------------------
------------------------------------------------------------
----------------------------------------+
| Warning | 1831 | Duplicate index 'full_name_1' defined on
the table 'employees.employees_index_example'. This is
deprecated and will be disallowed in a future release. |
+---------+------+--------------------------------------------
------------------------------------------------------------
----------------------------------------------------+
1 row in set (0.00 sec)

mysql> INSERT INTO employees_index_example SELECT
emp_no,birth_date,first_name,last_name,gender,hire_date FROM
employees;

mysql> RENAME TABLE employees TO employees_old;
mysql> RENAME TABLE employees_index_example TO employees;
```

Suppose you want to check whether any of `first_name` or `last_name` is `Adam`:

The explain plan is as follows:

```
mysql> EXPLAIN SELECT emp_no FROM employees WHERE
first_name='Adam' OR last_name='Adam'\G
*************************** 1. row
***************************
           id: 1
  select_type: SIMPLE
        table: employees
   partitions: NULL
         type: index_merge
possible_keys:
full_name,full_name_desc,first_name,full_name_1,first_name_e
mp_no,last_name_2
          key: first_name,last_name_2
      key_len: 58,42
          ref: NULL
         rows: 252
     filtered: 100.00
        Extra: Using sort_union(first_name,last_name_2);
Using where
1 row in set, 1 warning (0.00 sec)
```

You will notice that there are many options available for the optimizer to fulfill the query. It can use any of the indexes listed in `possible_keys: (full_name,full_name_desc,first_name,full_name_1,first_name_emp_no,last_name_2)`. The optimizer verifies all the plans and determines which plan involves the least cost.

Some examples of costs involved in the query are accessing data from the disk, accessing data from the memory, creating a temp table, sorting the results in the memory, and so on. MySQL assigns a relative value for each operation and sums the total cost for each plan. It executes the plan that involves the least cost.

How to do it...

You can control the optimizer either by passing hints to the query or by adjusting the variable at a global or session level. You can even adjust the cost of the operations. It is recommended to leave these values as default unless you know what you are doing.

optimizer_search_depth

The Jørgen's point of view, taken from http://jorgenloland.blogspot.in/2012/04/improvements-for-many-table-joins-in.html, states that:

"MySQL uses greedy search algorithm to to find the best order to join tables. When you join just a few tables, there's no problem calculating the cost of all join order combinations and then pick the best plan. However, since there are (#tables)! possible combinations, the cost of calculating them all soon becomes too high: for five tables, e.g., there are 120 combinations which is no problem to compute. For 10 tables there are 3.6 million combinations and for 15 tables there are 1307 billion. For this reason, MySQL makes a trade off: use heuristics to only explore promising plans. This is supposed to significantly reduce the number of plans MySQL needs to calculate, but at the same time you risk not finding the best one."

The MySQL documentation says:

"The optimizer_search_depth variable tells how far into the "future" of each incomplete plan the optimizer should look to evaluate whether it should be expanded further. Smaller values of optimizer_search_depth may result in orders of magnitude smaller query compilation times. For example, queries with 12, 13, or more tables may easily require hours and even days to compile if optimizer_search_depth is close to the number of tables in the query. At the same time, if compiled with optimizer_search_depth equal to 3 or 4, the optimizer may compile in less than a minute for the same query. If you are unsure of what a reasonable value is for

optimizer_search_depth, this variable can be set to 0 to tell the optimizer to determine the value automatically."

The default value of `optimizer_search_depth` is `62`, which is very greedy, but because of heuristics, MySQL picks up the plan very quickly. It is not clear from the documentation why the default value is set to `62` instead of `0`.

If you are joining more than seven tables, you can set `optimizer_search_depth` to `0` or pass the optimizer hint (you will learn that in the next section). Automatic selection picks the value of min (number of tables, seven), limiting the search depth to a reasonable value:

```
mysql> SHOW VARIABLES LIKE 'optimizer_search_depth';
+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| optimizer_search_depth | 62    |
+------------------------+-------+
1 row in set (0.00 sec)

mysql> SET @@SESSION.optimizer_search_depth=0;
Query OK, 0 rows affected (0.00 sec)
```

How to know that the query is spending time in evaluating plans?

If you are joining 10 tables (mostly autogenerated by ORM), run an explain plan. If it takes more time, it means that the query is spending too much time in evaluating plans. Adjust the value of `optimizer_search_depth` (probably set to `0`) and check how much time the explain plan takes. Also note down the change in plans when you adjust the value of `optimizer_search_depth`.

optimizer_switch

The `optimizer_switch` system variable is a set of flags. You can set each of those flags to `ON` or `OFF` to enable or disable the corresponding optimizer behavior. You can set it at the session level or global level dynamically. If you adjust the optimizer switch at the session level, all the queries in that session are affected, and if it is at the global level, all queries are affected.

For example, you have noticed that the preceding query, `SELECT emp_no FROM employees WHERE first_name='Adam' OR last_name='Adam'`, is using `sort_union(first_name,last_name_2)`. If you think that optimization is not correct for that query, you can adjust `optimizer_switch` to switch to another optimization:

```
mysql> SHOW VARIABLES LIKE 'optimizer_switch'\G
*************************** 1. row
***************************
Variable_name: optimizer_switch
        Value:
index_merge=on,index_merge_union=on,index_merge_sort_union=o
n,index_merge_intersection=on,engine_condition_pushdown=on,i
ndex_condition_pushdown=on,mrr=on,mrr_cost_based=on,block_ne
sted_loop=on,batched_key_access=off,materialization=on,semij
oin=on,loosescan=on,firstmatch=on,duplicateweedout=on,subque
ry_materialization_cost_based=on,use_index_extensions=on,con
dition_fanout_filter=on,derived_merge=on
1 row in set (0.00 sec)
```

Initially, `index_merge_union` is on:

```
mysql> EXPLAIN SELECT emp_no FROM employees WHERE
first_name='Adam' OR last_name='Adam'\G
*************************** 1. row
***************************
           id: 1
```

```
  select_type: SIMPLE
        table: employees
   partitions: NULL
         type: index_merge
possible_keys:
full_name,full_name_desc,first_name,full_name_1,first_name_e
mp_no,last_name_2
          key: first_name,last_name_2
      key_len: 58,42
          ref: NULL
         rows: 252
     filtered: 100.00
        Extra: Using sort_union(first_name,last_name_2);
Using where
1 row in set, 1 warning (0.00 sec)
```

The optimizer is able to use `sort_union`:

```
mysql> SET
@@SESSION.optimizer_switch="index_merge_sort_union=off";
Query OK, 0 rows affected (0.00 sec)
```

You can turn off `index_merge_sort_union` optimization at the session level so that only queries in this session are affected:

```
mysql>  SHOW VARIABLES LIKE 'optimizer_switch'\G
*************************** 1. row
***************************
Variable_name: optimizer_switch
        Value:
index_merge=on,index_merge_union=on,index_merge_sort_union=o
ff,index_merge_intersection=on,engine_condition_pushdown=on,
index_condition_pushdown=on,mrr=on,mrr_cost_based=on,block_n
ested_loop=on,batched_key_access=off,materialization=on,semi
join=on,loosescan=on,firstmatch=on,duplicateweedout=on,subqu
ery_materialization_cost_based=on,use_index_extensions=on,co
ndition_fanout_filter=on,derived_merge=on
1 row in set (0.00 sec)
```

You will notice the plan change after `index_merge_sort_union` is turned off; it is no longer using `sort_union` optimization:

```
mysql> EXPLAIN SELECT emp_no FROM employees WHERE
first_name='Adam' OR last_name='Adam'\G
*************************** 1. row
***************************
           id: 1
  select_type: SIMPLE
        table: employees
   partitions: NULL
         type: index
possible_keys:
full_name,full_name_desc,first_name,full_name_1,first_name_e
mp_no,last_name_2
          key: full_name
      key_len: 124
          ref: NULL
         rows: 299379
     filtered: 19.00
        Extra: Using where; Using index
1 row in set, 1 warning (0.00 sec)
```

You can further find that, in this case, using `sort_union` is the best choice. Refer to https://dev.mysql.com/doc/refman/8.0/en/switchable-optimizations.html for more details on all types of optimizer switches.

Optimizer hints

Instead of adjusting the optimizer switch or `optimizer_search_depth` variables at session level, you can hint the optimizer to use, or not to use, certain optimizations. The scope of the optimizer hint is limited to the statement that gives you finer control over the queries, whereas the optimizer switch can be at session or global level.

Again, take the example of the preceding query; if you feel that using `sort_union` is not optimal, you can turn it off by passing it as a hint in the query itself:

```
mysql> EXPLAIN SELECT /*+ NO_INDEX_MERGE(employees
first_name,last_name_2) */ * FROM employees WHERE
first_name='Adam' OR last_name='Adam'\G
*************************** 1. row
***************************
           id: 1
  select_type: SIMPLE
        table: employees
   partitions: NULL
         type: ALL
possible_keys:
full_name,full_name_desc,first_name,full_name_1,first_name_e
mp_no,last_name_2
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 299379
     filtered: 19.00
        Extra: Using where
1 row in set, 1 warning (0.00 sec)
```

Remember that in the redundant index section, we dropped the redundant index to find which plan was better. Instead, you can use the optimizer hint to ignore the intersect of `from_date` and `from_date_2`:

```
mysql> EXPLAIN SELECT /*+ NO_INDEX_MERGE(s
from_date,from_date_2) */ e.emp_no, salary FROM salaries s
JOIN employees e ON s.emp_no=e.emp_no WHERE from_date='2001-
05-23'\G
*************************** 1. row
***************************
           id: 1
  select_type: SIMPLE
        table: s
   partitions: NULL
         type: ref
possible_keys: PRIMARY,from_date,from_date_2
          key: from_date
      key_len: 3
          ref: const
         rows: 90
     filtered: 100.00
        Extra: NULL
*************************** 2. row
***************************
           id: 1
  select_type: SIMPLE
        table: e
   partitions: NULL
         type: ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: employees.s.emp_no
         rows: 1
     filtered: 100.00
        Extra: Using index
2 rows in set, 1 warning (0.00 sec)
```

Another good example of using optimizer hints is setting the `JOIN` order:

```
mysql> EXPLAIN SELECT e.emp_no, salary FROM salaries s JOIN
employees e ON s.emp_no=e.emp_no WHERE (first_name='Adam' OR
last_name='Adam') ORDER BY from_date DESC\G
*************************** 1. row
***************************
           id: 1
  select_type: SIMPLE
        table: e
   partitions: NULL
         type: index_merge
possible_keys:
PRIMARY,full_name,full_name_desc,first_name,full_name_1,firs
t_name_emp_no,last_name_2
          key: first_name,last_name_2
      key_len: 58,42
          ref: NULL
         rows: 252
     filtered: 100.00
        Extra: Using sort_union(first_name,last_name_2);
Using where; Using temporary; Using filesort
*************************** 2. row
***************************
           id: 1
  select_type: SIMPLE
        table: s
   partitions: NULL
         type: ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: employees.e.emp_no
         rows: 9
     filtered: 100.00
        Extra: NULL
2 rows in set, 1 warning (0.00 sec)
```

In the preceding query, the optimizer is first considering the `employees` table, and joining with the `salaries` table. You can change that by passing the hint, `/*+ JOIN_ORDER(s,e ) */`:

```
mysql> EXPLAIN SELECT /*+ JOIN_ORDER(s, e) */ e.emp_no,
salary FROM salaries s JOIN employees e ON s.emp_no=e.emp_no
WHERE (first_name='Adam' OR last_name='Adam') ORDER BY
from_date DESC\G
*************************** 1. row
```

```
***************************
           id: 1
  select_type: SIMPLE
        table: s
   partitions: NULL
         type: ALL
possible_keys: PRIMARY
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 2838426
     filtered: 100.00
        Extra: Using filesort
*************************** 2. row
***************************
           id: 1
  select_type: SIMPLE
        table: e
   partitions: NULL
         type: eq_ref
possible_keys:
PRIMARY,full_name,full_name_desc,first_name,full_name_1,firs
t_name_emp_no,last_name_2
          key: PRIMARY
      key_len: 4
          ref: employees.s.emp_no
         rows: 1
     filtered: 19.00
        Extra: Using where
2 rows in set, 1 warning (0.00 sec)
```

You will now notice that the `salaries` table is considered first, which avoids creating a temporary table, but it is going for a full table scan on the `salaries` table.

Another use case of optimizer hints is as follows: rather than setting the session variables for each statement or session, you can set them only for the statement. Suppose you are using an `ORDER BY` clause that sorts the query results, but you do not have index on the `ORDER BY` clause. Optimizer makes use of `sort_buffer_size` to speed up sorting. By default, the value of `sort_buffer_size` is `256K`. If `sort_buffer_size` is not sufficient, the number of merge passes that the sort algorithm has to do increases. You can measure this through the session variable `sort_merge_passes`:

```
mysql> SHOW SESSION status LIKE 'sort_merge_passes';
+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| Sort_merge_passes | 0     |
+-------------------+-------+
1 row in set (0.00 sec)

mysql> pager grep "rows in set"; SELECT * FROM employees
ORDER BY hire_date DESC;nopager;
PAGER set to 'grep "rows in set"'
300025 rows in set (0.45 sec)

PAGER set to stdout
mysql> SHOW SESSION status LIKE 'sort_merge_passes';
+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| Sort_merge_passes | 8     |
+-------------------+-------+
1 row in set (0.00 sec)
```

You will notice that MySQL did not have enough `sort_buffer_size`, and it has to do eight `sort_merge_passes`. You can set `sort_buffer_size` to some large value such as `16M` through an optimizer hint and check `sort_merge_passes`:

```
mysql> SHOW SESSION status LIKE 'sort_merge_passes';
+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| Sort_merge_passes | 0     |
+-------------------+-------+
1 row in set (0.00 sec)

mysql> pager grep "rows in set"; SELECT /*+
SET_VAR(sort_buffer_size = 16M) */ * FROM employees ORDER BY
```

```
hire_date DESC;nopager;
PAGER set to 'grep "rows in set"'
300025 rows in set (0.45 sec)

PAGER set to stdout
mysql> SHOW SESSION status LIKE 'sort_merge_passes';
+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| Sort_merge_passes | 0     |
+-------------------+-------+
1 row in set (0.00 sec)
```

You will notice that `sort_merge_passes` is `0` when `sort_buffer_size` is set to `16M`.

It is highly recommended to optimize your queries by using indexes rather than relying on `sort_buffer_size`. You can consider increasing the `sort_buffer_size` value to speed up `ORDER BY` or `GROUP BY` operations that cannot be improved with query optimization or improved indexing.

Using `SET_VAR`, you can set `optimizer_switch` at the statement level:

```
mysql> EXPLAIN SELECT /*+ SET_VAR(optimizer_switch =
'index_merge_sort_union=off') */ e.emp_no, salary FROM
salaries s JOIN employees e ON s.emp_no=e.emp_no WHERE
from_date='2001-05-23'\G
*************************** 1. row
***************************
           id: 1
  select_type: SIMPLE
        table: e
   partitions: NULL
         type: index
possible_keys: PRIMARY
          key: name
      key_len: 124
          ref: NULL
         rows: 299379
     filtered: 100.00
```

```
        Extra: Using index
*************************** 2. row
***************************
           id: 1
  select_type: SIMPLE
        table: s
   partitions: NULL
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 7
          ref: employees.e.emp_no,const
         rows: 1
     filtered: 100.00
        Extra: NULL
2 rows in set, 1 warning (0.00 sec)
```

You can also set the maximum execution time for a query, meaning the query is automatically terminated after the specified time using `/*+ MAX_EXECUTION_TIME(milli seconds) */`:

```
mysql> SELECT /*+ MAX_EXECUTION_TIME(100) */ * FROM
employees ORDER BY hire_date DESC;
ERROR 1028 (HY000): Sort aborted: Query execution was
interrupted, maximum statement execution time exceeded
```

You can hint many other things to the optimizer, refer to https://dev.mysql.com/doc/refman/8.0/en/optimizer-hints.html to get complete list and more examples.

Adjusting the optimizer cost model

To generate execution plans, the optimizer uses a cost model that is based on estimates of the cost of various operations that occur during query execution. The optimizer has a set of compiled-in default cost constants available to it, to make decisions regarding execution plans. You can adjust them by updating or inserting the `mysql.engine_cost` table and executing the `FLUSH OPTIMIZER_COSTS` command:

```
mysql> SELECT * FROM mysql.engine_cost\G
*************************** 1. row
***************************
  engine_name: InnoDB
  device_type: 0
    cost_name: io_block_read_cost
   cost_value: 1
  last_update: 2017-11-20 16:24:56
      comment: NULL
default_value: 1
*************************** 2. row
***************************
  engine_name: InnoDB
  device_type: 0
    cost_name: memory_block_read_cost
   cost_value: 0.25
  last_update: 2017-11-19 13:58:32
      comment: NULL
default_value: 0.25
2 rows in set (0.00 sec)
```

Suppose you have a superfast disk; you can decrease the `cost_value` for `io_block_read_cost`:

```
mysql> UPDATE mysql.engine_cost SET cost_value=0.5 WHERE
cost_name='io_block_read_cost';
Query OK, 1 row affected (0.08 sec)
```

```
Rows matched: 1  Changed: 1  Warnings: 0

mysql> FLUSH OPTIMIZER_COSTS;
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT * FROM mysql.engine_cost\G
*************************** 1. row
***************************
  engine_name: InnoDB
  device_type: 0
    cost_name: io_block_read_cost
   cost_value: 0.5
  last_update: 2017-11-20 17:02:43
      comment: NULL
default_value: 1
*************************** 2. row
***************************
  engine_name: InnoDB
  device_type: 0
    cost_name: memory_block_read_cost
   cost_value: 0.25
  last_update: 2017-11-19 13:58:32
      comment: NULL
default_value: 0.25
2 rows in set (0.00 sec)
```

To know more about the optimizer cost model, refer to https://dev.mysql.com/doc/refman/8.0/en/cost-model.html.

Using index hints

Using index hints, you can hint the optimizer to use or ignore indexes. This is different from optimizer hints. In optimizer hints, you hint the optimizer to use or ignore certain optimization methods. Index and optimizer hints can be used separately or together to achieve the desired plan. Index hints are specified following a table name.

When you are executing a complex query involving multiple table joins, and if the optimizer is taking too much time in evaluating the plans, you can determine the best plan and give it a hint to the query. But make sure that the plan you are suggesting is the best and should work in all cases.

How to do it...

Take the same query where you evaluated the use of the redundant index as an example; it is using `intersect(from_date,from_date_2)`. By passing the optimizer hint `(/*+ NO_INDEX_MERGE(s from_date,from_date_2) */)`, you avoided the use of intersect. You can achieve the same behavior by hinting the optimizer to ignore the `from_date_2` index:

```
mysql> EXPLAIN SELECT e.emp_no, salary FROM salaries s
IGNORE INDEX(from_date_2) JOIN employees e ON
s.emp_no=e.emp_no WHERE from_date='2001-05-23'\G
*************************** 1. row
***************************
           id: 1
  select_type: SIMPLE
        table: s
   partitions: NULL
         type: ref
possible_keys: PRIMARY,from_date
          key: from_date
      key_len: 3
          ref: const
         rows: 90
     filtered: 100.00
        Extra: NULL
*************************** 2. row
***************************
           id: 1
  select_type: SIMPLE
        table: e
   partitions: NULL
         type: ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: employees.s.emp_no
         rows: 1
     filtered: 100.00
```

```
        Extra: Using index
2 rows in set, 1 warning (0.00 sec)
```

Another use case is hinting the optimizer and saving the cost of evaluating multiple plans. Consider the following `employees` table and the query (the same as the one discussed at the beginning of the *Controlling query optimizer* section):

```
mysql> SHOW CREATE TABLE employees\G
*************************** 1. row
***************************
       Table: employees
Create Table: CREATE TABLE `employees` (
  `emp_no` int(11) NOT NULL,
  `birth_date` date NOT NULL,
  `first_name` varchar(14) NOT NULL,
  `last_name` varchar(16) NOT NULL,
  `gender` enum('M','F') NOT NULL,
  `hire_date` date NOT NULL,
  PRIMARY KEY (`emp_no`),
  KEY `last_name` (`last_name`) /*!80000 INVISIBLE */,
  KEY `full_name` (`first_name`,`last_name`),
  KEY `full_name_desc` (`first_name` DESC,`last_name`),
  KEY `first_name` (`first_name`),
  KEY `full_name_1` (`first_name`,`last_name`),
  KEY `first_name_emp_no` (`first_name`,`emp_no`),
  KEY `last_name_2` (`last_name`(10))
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
1 row in set (0.00 sec)


mysql> EXPLAIN SELECT emp_no FROM employees WHERE
first_name='Adam' OR last_name='Adam'\G
*************************** 1. row
***************************
           id: 1
  select_type: SIMPLE
        table: employees
   partitions: NULL
         type: index_merge
possible_keys:
full_name,full_name_desc,first_name,full_name_1,first_name_e
mp_no,last_name_2
```

```
           key: first_name,last_name_2
       key_len: 58,42
           ref: NULL
          rows: 252
      filtered: 100.00
         Extra: Using sort_union(first_name,last_name_2);
Using where
1 row in set, 1 warning (0.00 sec)
```

You can see that the optimizer has to evaluate the indexes `full_name`, `full_name_desc`, `first_name`, `full_name_1`, `first_name_emp_no`, `last_name_2` to arrive at the best plan. You can hint the optimizer by passing `USE INDEX(first_name,last_name_2)`, which will eliminate scanning of other indexes:

```
mysql> EXPLAIN SELECT emp_no FROM employees USE
INDEX(first_name,last_name_2) WHERE first_name='Adam' OR
last_name='Adam'\G
*************************** 1. row
***************************
           id: 1
  select_type: SIMPLE
        table: employees
   partitions: NULL
         type: index_merge
possible_keys: first_name,last_name_2
          key: first_name,last_name_2
      key_len: 58,42
          ref: NULL
         rows: 252
     filtered: 100.00
        Extra: Using sort_union(first_name,last_name_2);
Using where
1 row in set, 1 warning (0.00 sec)
```

Since this is a simple query and the table is very small, the performance gain is negligible. The performance gain can be significant when the query is complex and is executed millions of times an hour.

Indexing for JSON using generated columns

JSON columns cannot be indexed directly. So if you want to use an index on a JSON column, you can extract the information using virtual columns and a created index on the virtual column.

How to do it...

1. Consider the `emp_details` table that you created in Chapter 3, *Using MySQL (Advanced), Using JSON* section:

```
mysql> SHOW CREATE TABLE emp_details\G
*************************** 1. row
***************************
       Table: emp_details
Create Table: CREATE TABLE `emp_details` (
  `emp_no` int(11) NOT NULL,
  `details` json DEFAULT NULL,
  PRIMARY KEY (`emp_no`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
1 row in set (0.00 sec)
```

2. Insert a few dummy records:

```
mysql> INSERT IGNORE INTO emp_details(emp_no, details)
VALUES
     ('1', '{ "location": "IN", "phone":
"+11800000000", "email": "abc@example.com", "address":
{ "line1": "abc", "line2": "xyz street", "city":
"Bangalore", "pin": "560103"}}'),
     ('2', '{ "location": "IN", "phone":
"+11800000000", "email": "def@example.com", "address":
{ "line1": "abc", "line2": "xyz street", "city":
"Delhi", "pin": "560103"}}'),
     ('3', '{ "location": "IN", "phone":
"+11800000000", "email": "ghi@example.com", "address":
{ "line1": "abc", "line2": "xyz street", "city":
"Mumbai", "pin": "560103"}}'),
     ('4', '{ "location": "IN", "phone":
"+11800000000", "email": "jkl@example.com", "address":
{ "line1": "abc", "line2": "xyz street", "city":
"Delhi", "pin": "560103"}}'),
     ('5', '{ "location": "US", "phone":
"+11800000000", "email": "mno@example.com", "address":
{ "line1": "abc", "line2": "xyz street", "city":
"Sunnyvale", "pin": "560103"}}');
```

```
Query OK, 5 rows affected (0.00 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

3. Suppose you want to retrieve the `emp_no` whose city is `Bangalore`:

```
mysql> EXPLAIN SELECT emp_no FROM emp_details WHERE
details->>'$.address.city'="Bangalore"\G
*************************** 1. row
***************************
           id: 1
  select_type: SIMPLE
        table: emp_details
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 5
     filtered: 100.00
        Extra: Using where
1 row in set, 1 warning (0.00 sec)
```

   You will notice that the query is not able to use the index and scan all the rows.

4. You can retrieve the city as a virtual column and add an index on it:

```
mysql> ALTER TABLE emp_details ADD COLUMN city
varchar(20) AS (details->>'$.address.city'), ADD INDEX
(city);
Query OK, 0 rows affected (0.22 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW CREATE TABLE emp_details\G
*************************** 1. row
***************************
       Table: emp_details
Create Table: CREATE TABLE `emp_details` (
  `emp_no` int(11) NOT NULL,
  `details` json DEFAULT NULL,
  `city` varchar(20) GENERATED ALWAYS AS
(json_unquote(json_extract(`details`,_utf8'$.address.ci
ty'))) VIRTUAL,
  PRIMARY KEY (`emp_no`),
  KEY `city` (`city`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
1 row in set (0.01 sec)
```

5. If you check the explain plan now, you can notice that the query is able to use the index on `city` and scan only one row:

```
mysql> EXPLAIN SELECT emp_no FROM emp_details WHERE
details->>'$.address.city'="Bangalore"\G
*************************** 1. row
***************************
           id: 1
  select_type: SIMPLE
        table: emp_details
   partitions: NULL
         type: ref
possible_keys: city
          key: city
      key_len: 83
          ref: const
         rows: 1
     filtered: 100.00
        Extra: NULL
1 row in set, 1 warning (0.00 sec)
```

To know more about secondary indexes on generated columns, refer to https://dev.mysql.com/doc/refman/8.0/en/create-table-secondary-indexes.html.

Using resource groups

You can restrict the queries to use only a certain number of system resources using the resource groups. Currently, only CPU time is a manageable resource represented by **virtual CPU** (**VCPU**), which includes CPU cores, hyperthreads, hardware threads, and more. You can create a resource group and assign the VCPUs to it. Apart from the CPU, the attribute to the resource group is thread priority.

You can assign a resource group to a thread, set the default resource group at the session level, or pass the resource group as an optimizer hint. For example, you want to run some queries (say, reporting queries) with lowest priority; you can assign them to a resource group that has minimum resources.

How to do it...

1. Set the `CAP_SYS_NICE` capability to `mysqld`:

```
shell> ps aux | grep mysqld | grep -v grep
mysql     5238  0.0 28.1 1253368 488472 ?      Sl
Nov19   4:04 /usr/sbin/mysqld --daemonize --pid-
file=/var/run/mysqld/mysqld.pid

shell> sudo setcap cap_sys_nice+ep /usr/sbin/mysqld

shell> getcap /usr/sbin/mysqld
/usr/sbin/mysqld = cap_sys_nice+ep
```

2. Create a resource group using the `CREATE RESOURCE GROUP` statement. You have to mention the resource group name, number of VCPUS, thread priority, and type, which can be either `USER` or `SYSTEM`. If you do not specify the VCPU, all the CPUs will be used:

```
mysql> CREATE RESOURCE GROUP report_group
TYPE = USER
VCPU = 2-3
THREAD_PRIORITY = 15
ENABLE;
# You should have at least 4 CPUs for the above
resource group to create. If you have less CPUs, you
can use VCPU = 0-1 for testing the example.
```

The VCPU represents the CPU number as 0-5, including CPUs 0, 1, 2, 3, 4, and 5; and 0-3, 8-9, and 11 include CPUs 0, 1, 2, 3, 8, 9, and 11.

The `THREAD_PRIORITY` is like a nice value for the CPU; it ranges from -20 to 0 for system resource groups and 0 to 19 for user groups. -20 is the highest priority and 19 is the lowest priority.

You can also enable or disable a resource group. By default, the resource group is enabled at creation. A disabled group cannot have threads assigned to it.

3. After creating, you can verify the resource groups created:

```
mysql> SELECT * FROM
INFORMATION_SCHEMA.RESOURCE_GROUPS\G
*************************** 1. row
***************************
   RESOURCE_GROUP_NAME: USR_default
   RESOURCE_GROUP_TYPE: USER
RESOURCE_GROUP_ENABLED: 1
              VCPU_IDS: 0-0
       THREAD_PRIORITY: 0
*************************** 2. row
***************************
   RESOURCE_GROUP_NAME: SYS_default
   RESOURCE_GROUP_TYPE: SYSTEM
RESOURCE_GROUP_ENABLED: 1
              VCPU_IDS: 0-0
       THREAD_PRIORITY: 0
*************************** 3. row
***************************
   RESOURCE_GROUP_NAME: report_group
   RESOURCE_GROUP_TYPE: USER
RESOURCE_GROUP_ENABLED: 1
              VCPU_IDS: 2-3
       THREAD_PRIORITY: 15
```

`USR_default` and `SYS_default` are default resource groups which cannot be dropped or modified.

4. Assign a group to a thread:

```
mysql> SET RESOURCE GROUP report_group FOR <thread_id>;
```

5. Set the sessions resource group; all the queries in that session will be executed under `report_group`:

```
mysql> SET RESOURCE GROUP report_group;
```

6. Use the `RESOURCE_GROUP` optimizer hint to execute a single statement using `report_group`:

```
mysql> SELECT /*+ RESOURCE_GROUP(report_group) */ *
FROM employees;
```

Alter and drop resource group

You can dynamically adjust the number of CPUs or `thread_priority` of a resource group. If the system is heavily loaded, you can decrease the thread priority:

```
mysql> ALTER RESOURCE GROUP report_group VCPU = 3
THREAD_PRIORITY = 19;
Query OK, 0 rows affected (0.12 sec)
```

Similarly, you can increase the priority when the system is lightly loaded:

```
mysql> ALTER RESOURCE GROUP report_group VCPU = 0-12
THREAD_PRIORITY = 0;
Query OK, 0 rows affected (0.12 sec)
```

You can disable a resource group:

```
mysql> ALTER RESOURCE GROUP report_group DISABLE FORCE;
Query OK, 0 rows affected (0.00 sec)
```

You can also drop a resource group using the `DROP RESOURCE GROUP` statement:

```
mysql> DROP RESOURCE GROUP report_group FORCE;
```

If `FORCE` is given, the threads running are moved to the default resource group (system threads to `SYS_default` and user threads to `USR_default`).

If `FORCE` is not given, existing threads in the group continue to run until they terminate, but new threads cannot be assigned to the group.

The resource group is restricted to the local server, and none of the resource-group-related statements are replicated. To know more about resource groups, refer to https://dev.mysql.com/doc/refman/8.0/en/resource-groups.html.

Using performance_schema

You can inspect the internal execution of the server at runtime using `performance_schema`. This should not be confused with information schema, which is used to inspect metadata.

There are many event consumers in `performance_schema` that influence the timings of a server, such as a function call, a wait for the operating system, a stage of an SQL statement execution (say, parsing or sorting), a single statement, or a group of statements. All the collected information is stored in `performance_schema` and is not replicated.

`performance_schema` is enabled by default; if you want to disable it, you can set `performance_schema=OFF` in the `my.cnf` file. By default, not all the consumers and instruments are enabled; you can turn them off/on by updating the `performance_schema.setup_instruments` and `performance_schema.setup_consumers` tables.

How to do it...

We will see how to use the `performance_schema`.

Enable/disable performance_schema

To disable it, set `performance_schema` to `0`:

```
shell> sudo vi /etc/my.cnf
[mysqld]
performance_schema = 0
```

Enable/disable consumers and instruments

You see a list of consumers available in the `setup_consumers` table, as follows:

```
mysql> SELECT * FROM performance_schema.setup_consumers;
+----------------------------------+---------+
| NAME                             | ENABLED |
+----------------------------------+---------+
| events_stages_current            | NO      |
| events_stages_history            | NO      |
| events_stages_history_long       | NO      |
| events_statements_current        | YES     |
| events_statements_history        | YES     |
| events_statements_history_long   | NO      |
| events_transactions_current      | YES     |
| events_transactions_history      | YES     |
| events_transactions_history_long | NO      |
| events_waits_current             | NO      |
| events_waits_history             | NO      |
| events_waits_history_long        | NO      |
| global_instrumentation           | YES     |
| thread_instrumentation           | YES     |
| statements_digest                | YES     |
+----------------------------------+---------+
15 rows in set (0.00 sec)
```

Suppose you want to enable `events_waits_current`:

```
mysql> UPDATE performance_schema.setup_consumers SET
ENABLED='YES' WHERE NAME='events_waits_current';
```

Similarly, you can disable or enable instruments from the `setup_instruments` table. There are around 1182 instruments (depending on the version):

```
mysql> SELECT NAME, ENABLED, TIMED FROM setup_instruments
LIMIT 10;
+---------------------------------------------------------+-
--------+-------+
| NAME                                                    |
ENABLED | TIMED |
+---------------------------------------------------------+-
--------+-------+
| wait/synch/mutex/pfs/LOCK_pfs_share_list                |
NO      | NO    |
| wait/synch/mutex/sql/TC_LOG_MMAP::LOCK_tc               |
NO      | NO    |
| wait/synch/mutex/sql/MYSQL_BIN_LOG::LOCK_commit         |
NO      | NO    |
| wait/synch/mutex/sql/MYSQL_BIN_LOG::LOCK_commit_queue   |
NO      | NO    |
| wait/synch/mutex/sql/MYSQL_BIN_LOG::LOCK_done           |
NO      | NO    |
| wait/synch/mutex/sql/MYSQL_BIN_LOG::LOCK_flush_queue    |
NO      | NO    |
| wait/synch/mutex/sql/MYSQL_BIN_LOG::LOCK_index          |
NO      | NO    |
| wait/synch/mutex/sql/MYSQL_BIN_LOG::LOCK_log            |
NO      | NO    |
| wait/synch/mutex/sql/MYSQL_BIN_LOG::LOCK_binlog_end_pos |
NO      | NO    |
| wait/synch/mutex/sql/MYSQL_BIN_LOG::LOCK_sync           |
NO      | NO    |
+---------------------------------------------------------+-
--------+-------+
10 rows in set (0.00 sec)
```

performance_schema tables

There are five main types of table in `performance_schema`. They are current events tables, event history tables, event summary tables, object instances tables, and setup (configuration) tables:

```
mysql> SHOW TABLES LIKE '%current%';
+------------------------------------------+
| Tables_in_performance_schema (%current%) |
+------------------------------------------+
| events_stages_current                    |
| events_statements_current                |
| events_transactions_current              |
| events_waits_current                     |
+------------------------------------------+
4 rows in set (0.00 sec)

mysql> SHOW TABLES LIKE '%history%';
+------------------------------------------+
| Tables_in_performance_schema (%history%) |
+------------------------------------------+
| events_stages_history                    |
| events_stages_history_long               |
| events_statements_history                |
| events_statements_history_long           |
| events_transactions_history              |
| events_transactions_history_long         |
| events_waits_history                     |
| events_waits_history_long                |
+------------------------------------------+
8 rows in set (0.00 sec)

mysql> SHOW TABLES LIKE '%summary%';
+------------------------------------------------------+
| Tables_in_performance_schema (%summary%)             |
+------------------------------------------------------+
| events_errors_summary_by_account_by_error            |
| events_errors_summary_by_host_by_error               |
~
~
```

```
| table_io_waits_summary_by_table                      |
| table_lock_waits_summary_by_table                    |
+------------------------------------------------------+
41 rows in set (0.00 sec)

mysql> SHOW TABLES LIKE '%setup%';
+----------------------------------------+
| Tables_in_performance_schema (%setup%) |
+----------------------------------------+
| setup_actors                           |
| setup_consumers                        |
| setup_instruments                      |
| setup_objects                          |
| setup_threads                          |
| setup_timers                           |
+----------------------------------------+
6 rows in set (0.00 sec)
```

Suppose you want to find out which file is accessed the most:

```
mysql> SELECT EVENT_NAME, COUNT_STAR from
file_summary_by_event_name ORDER BY count_star DESC LIMIT
10;
+--------------------------------------------------+---------
---+
| EVENT_NAME                                       |
COUNT_STAR |
+--------------------------------------------------+---------
---+
| wait/io/file/innodb/innodb_data_file             |
35014 |
| wait/io/file/sql/io_cache                        |
13454 |
| wait/io/file/sql/binlog                          |
8785 |
| wait/io/file/innodb/innodb_log_file              |
2070 |
| wait/io/file/sql/query_log                       |
1257 |
| wait/io/file/innodb/innodb_temp_file             |
96 |
```

```
| wait/io/file/innodb/innodb_tablespace_open_file |
88 |
| wait/io/file/sql/casetest                         |
15 |
| wait/io/file/sql/binlog_index                     |
14 |
| wait/io/file/mysys/cnf                            |
5 |
+--------------------------------------------------+---------
---+
10 rows in set (0.00 sec)
```

Or you want to find out which file has taken the most time in writing:

```
mysql> SELECT EVENT_NAME, SUM_TIMER_WRITE FROM
file_summary_by_event_name ORDER BY SUM_TIMER_WRITE DESC
LIMIT 10;
+--------------------------------------------------+---------
--------+
| EVENT_NAME                                        |
SUM_TIMER_WRITE |
+--------------------------------------------------+---------
--------+
| wait/io/file/innodb/innodb_data_file              |
410909759715 |
| wait/io/file/innodb/innodb_log_file               |
366157166830 |
| wait/io/file/sql/io_cache                         |
341899621700 |
| wait/io/file/sql/query_log                        |
203975010330 |
| wait/io/file/sql/binlog                           |
85261691515 |
| wait/io/file/innodb/innodb_temp_file              |
25291378385 |
| wait/io/file/innodb/innodb_tablespace_open_file |
674778195 |
| wait/io/file/sql/SDI                              |
18981690 |
| wait/io/file/sql/pid                              |
10233405 |
| wait/io/file/archive/FRM                          |
0 |
+--------------------------------------------------+---------
--------+
```

You can use the `events_statements_summary_by_digest` table to get the query report, just like you did for `pt-query-digest`. Top query by amount of time taken:

```
mysql> SELECT SCHEMA_NAME, digest, digest_text,
round(sum_timer_wait/ 1000000000000, 6) as avg_time,
count_star FROM
performance_schema.events_statements_summary_by_digest ORDER
BY sum_timer_wait DESC LIMIT 1\G
*************************** 1. row
***************************
SCHEMA_NAME: NULL
     digest: 719f469393f90c27d84681a1d0ab3c19
digest_text: SELECT `sleep` (?)
   avg_time: 60.000442
 count_star: 1
1 row in set (0.00 sec)
```

Top query by number of executions:

```
mysql> SELECT SCHEMA_NAME, digest, digest_text,
round(sum_timer_wait/ 1000000000000, 6) as avg_time,
count_star FROM
performance_schema.events_statements_summary_by_digest ORDER
BY count_star DESC LIMIT 1\G
*************************** 1. row
***************************
SCHEMA_NAME: employees
     digest: f5296ec6642c0fb977b448b350a2ba9b
digest_text: INSERT INTO `salaries` VALUES (...) /* , ... */
   avg_time: 32.736742
 count_star: 114
1 row in set (0.01 sec)
```

Suppose you want to find the statistics of a particular query; rather than depending on `mysqlslap` benchmark, you can check all the statistics using `performance_schema`:

```
mysql> SELECT * FROM events_statements_summary_by_digest
WHERE DIGEST_TEXT LIKE '%SELECT%employee%ORDER%' LIMIT 1\G
*************************** 1. row
```

```
***************************
                SCHEMA_NAME: employees
                     DIGEST:
d3b56f71f362f1bf6b067bfa358c04ab
                DIGEST_TEXT: EXPLAIN SELECT /*+ SET_VAR (
`sort_buffer_size` = ? ) */ `e` . `emp_no` , `salary` FROM
`salaries` `s` JOIN `employees` `e` ON `s` . `emp_no` = `e`
. `emp_no` WHERE ( `first_name` = ? OR `last_name` = ? )
ORDER BY `from_date` DESC
                 COUNT_STAR: 1
             SUM_TIMER_WAIT: 643710000
             MIN_TIMER_WAIT: 643710000
             AVG_TIMER_WAIT: 643710000
             MAX_TIMER_WAIT: 643710000
              SUM_LOCK_TIME: 288000000
                 SUM_ERRORS: 0
               SUM_WARNINGS: 1
          SUM_ROWS_AFFECTED: 0
              SUM_ROWS_SENT: 2
          SUM_ROWS_EXAMINED: 0
SUM_CREATED_TMP_DISK_TABLES: 0
     SUM_CREATED_TMP_TABLES: 0
       SUM_SELECT_FULL_JOIN: 0
~
                 FIRST_SEEN: 2017-11-23 08:40:28.565406
                  LAST_SEEN: 2017-11-23 08:40:28.565406
                QUANTILE_95: 301995172
                QUANTILE_99: 301995172
               QUANTILE_999: 301995172
          QUERY_SAMPLE_TEXT: EXPLAIN SELECT /*+
SET_VAR(sort_buffer_size = 16M) */ e.emp_no, salary FROM
salaries s JOIN employees e ON s.emp_no=e.emp_no WHERE
(first_name='Adam' OR last_name='Adam') ORDER BY from_date
DESC
          QUERY_SAMPLE_SEEN: 2017-11-23 08:40:28.565406
    QUERY_SAMPLE_TIMER_WAIT: 643710000
```

Using the sys schema

The `sys` schema helps you interpret the data collected from the `performance_schema` in an easy and more understandable form. `performance_schema` should be enabled for `sys` schema to work. To use the `sys` schema to its fullest extent, you need to enable all the consumers and timers on `performance_schema`, but this impacts the performance of the server. So, enable consumers for only those that you are looking for.

A view with the `x$` prefix displays data in picoseconds, which is used by other tools for further processing; other tables are human readable.

How to do it...

Enable a instrument from the `sys` schema:

```
mysql> CALL sys.ps_setup_enable_instrument('statement');
+------------------------+
| summary                |
+------------------------+
| Enabled 22 instruments |
+------------------------+
1 row in set (0.08 sec)

Query OK, 0 rows affected (0.08 sec)
```

If you want to reset to default, do this:

```
mysql> CALL sys.ps_setup_reset_to_default(TRUE)\G
*************************** 1. row
***************************
status: Resetting: setup_actors
DELETE FROM performance_schema.setup_actors WHERE NOT (HOST
= '%' AND USER = '%' AND `ROLE` = '%')
1 row in set (0.01 sec)
~
*************************** 1. row
***************************
status: Resetting: threads
UPDATE performance_schema.threads SET INSTRUMENTED = 'YES'
1 row in set (0.03 sec)

Query OK, 0 rows affected (0.03 sec)
```

There are numerous tables in the `sys` schema; some of the most used ones are shown in this section.

Statement by type (INSERT and SELECT) from each host

```
mysql> SELECT statement, total, total_latency, rows_sent,
rows_examined, rows_affected, full_scans FROM
sys.host_summary_by_statement_type WHERE host='localhost'
ORDER BY total DESC LIMIT 5;
+------------+--------+--------------+----------+--------
------+--------------+-----------+
| statement  | total  | total_latency | rows_sent |
rows_examined | rows_affected | full_scans |
+------------+--------+--------------+----------+--------
------+--------------+-----------+
| select     | 208526 | 1.14 d        |  27484761 |
799220003 |             0 |      9265 |
| Quit       | 199551 | 4.76 s        |         0 |
0 |             0 |         0 |
| insert     |   9848 | 12.75 m       |         0 |
0 |       5075058 |         0 |
| Ping       |   4674 | 278.76 ms     |         0 |
0 |             0 |         0 |
| set_option |   2552 | 634.76 ms     |         0 |
0 |             0 |         0 |
+------------+--------+--------------+----------+--------
------+--------------+-----------+
6 rows in set (0.00 sec)
```

Statement by type from each user

```
mysql> SELECT statement, total, total_latency, rows_sent,
rows_examined, rows_affected, full_scans FROM
sys.user_summary_by_statement_type ORDER BY total DESC LIMIT
5;
+------------+--------+---------------+-----------+--------
------+---------------+-----------+
| statement  | total  | total_latency | rows_sent |
rows_examined | rows_affected | full_scans |
+------------+--------+---------------+-----------+--------
------+---------------+-----------+
| select     | 208535 | 1.14 d        |  27485256 |
799246972 |             0 |      9273 |
| Quit       | 199551 | 4.76 s        |         0 |
0 |             0 |         0 |
| insert     |   9848 | 12.75 m       |         0 |
0 |       5075058 |         0 |
| Ping       |   4674 | 278.76 ms     |         0 |
0 |             0 |         0 |
| set_option |   2552 | 634.76 ms     |         0 |
0 |             0 |         0 |
+------------+--------+---------------+-----------+--------
------+---------------+-----------+
5 rows in set (0.01 sec)
```

Redundant indexes

```
mysql> SELECT * FROM sys.schema_redundant_indexes WHERE
table_name='employees'\G
*************************** 1. row
***************************
              table_schema: employees
                table_name: employees
      redundant_index_name: first_name
   redundant_index_columns: first_name
redundant_index_non_unique: 1
       dominant_index_name: first_name_emp_no
    dominant_index_columns: first_name,emp_no
 dominant_index_non_unique: 1
            subpart_exists: 0
            sql_drop_index: ALTER TABLE
`employees`.`employees` DROP INDEX `first_name`
~
*************************** 8. row
***************************
              table_schema: employees
                table_name: employees
      redundant_index_name: last_name_2
   redundant_index_columns: last_name
redundant_index_non_unique: 1
       dominant_index_name: last_name
    dominant_index_columns: last_name
 dominant_index_non_unique: 1
            subpart_exists: 1
            sql_drop_index: ALTER TABLE
`employees`.`employees` DROP INDEX `last_name_2`
8 rows in set (0.00 sec)
```

Unused indexes

```
mysql> SELECT * FROM sys.schema_unused_indexes WHERE
object_schema='employees';
+---------------+----------------+-------------------+
| object_schema | object_name    | index_name        |
+---------------+----------------+-------------------+
| employees     | departments    | dept_name         |
| employees     | dept_emp       | dept_no           |
| employees     | dept_manager   | dept_no           |
| employees     | employees      | name              |
| employees     | employees1     | last_name         |
| employees     | employees1     | full_name         |
| employees     | employees1     | full_name_desc    |
| employees     | employees1     | first_name        |
| employees     | employees1     | full_name_1       |
| employees     | employees1     | first_name_emp_no |
| employees     | employees1     | last_name_2       |
| employees     | employees_mgr  | manager_id        |
| employees     | employees_test | name              |
| employees     | emp_details    | city              |
+---------------+----------------+-------------------+
14 rows in set (0.00 sec)
```

Statements executed from each host

```
mysql> SELECT * FROM sys.host_summary ORDER BY statements
DESC LIMIT 1\G
*************************** 1. row
***************************
                  host: localhost
            statements: 431214
     statement_latency: 1.15 d
 statement_avg_latency: 231.14 ms
           table_scans: 9424
              file_ios: 671972
       file_io_latency: 4.13 m
   current_connections: 3
     total_connections: 200193
          unique_users: 1
        current_memory: 0 bytes
total_memory_allocated: 0 bytes
1 row in set (0.02 sec)
```

Table statistics

```
mysql> SELECT * FROM sys.schema_table_statistics LIMIT 1\G
*************************** 1. row
***************************
     table_schema: employees
       table_name: employees
    total_latency: 14.03 h
     rows_fetched: 731760045
    fetch_latency: 14.03 h
    rows_inserted: 300025
   insert_latency: 2.81 s
     rows_updated: 0
   update_latency: 0 ps
     rows_deleted: 0
   delete_latency: 0 ps
 io_read_requests: NULL
          io_read: NULL
  io_read_latency: NULL
io_write_requests: NULL
         io_write: NULL
 io_write_latency: NULL
 io_misc_requests: NULL
  io_misc_latency: NULL
1 row in set (0.01 sec)
```

Table statistics with buffer

```
mysql> SELECT * FROM sys.schema_table_statistics_with_buffer
LIMIT 1\G
*************************** 1. row
***************************
              table_schema: employees
                table_name: employees
              rows_fetched: 731760045
             fetch_latency: 14.03 h
             rows_inserted: 300025
            insert_latency: 2.81 s
              rows_updated: 0
            update_latency: 0 ps
              rows_deleted: 0
            delete_latency: 0 ps
~
   innodb_buffer_allocated: 6.80 MiB
        innodb_buffer_data: 6.23 MiB
        innodb_buffer_free: 582.77 KiB
       innodb_buffer_pages: 435
innodb_buffer_pages_hashed: 0
   innodb_buffer_pages_old: 435
 innodb_buffer_rows_cached: 147734
1 row in set (0.13 sec)
```

Statement analysis

This output is similar to the output of `performance_schema.events_statements_summary_by_digest` and `pt-query-digest`.

The top query by execution count is as follows:

```
mysql> SELECT * FROM sys.statement_analysis ORDER BY
exec_count DESC LIMIT 1\G
*************************** 1. row
***************************
            query: SELECT `e` . `emp_no` , `salar ...
emp_no` WHERE `from_date` = ?
               db: employees
        full_scan:
       exec_count: 159997
        err_count: 0
       warn_count: 0
    total_latency: 1.98 h
      max_latency: 661.58 ms
      avg_latency: 44.54 ms
     lock_latency: 1.28 m
        rows_sent: 14400270
    rows_sent_avg: 90
    rows_examined: 28800540
rows_examined_avg: 180
    rows_affected: 0
rows_affected_avg: 0
       tmp_tables: 0
  tmp_disk_tables: 0
      rows_sorted: 0
sort_merge_passes: 0
           digest: 94c925c0f00e06566d0447822066b1fe
       first_seen: 2017-11-23 05:39:09
        last_seen: 2017-11-23 05:45:45
1 row in set (0.01 sec)
```

The statement that consumed maximum `tmp_disk_tables`:

```
mysql> SELECT * FROM sys.statement_analysis ORDER BY
tmp_disk_tables DESC LIMIT 1\G
*************************** 1. row
***************************
            query: SELECT `cat` . `name` AS `TABL ... SE
`col` . `type` WHEN ? THEN
               db: employees
        full_scan:
       exec_count: 195
        err_count: 0
       warn_count: 0
    total_latency: 249.55 ms
      max_latency: 2.84 ms
      avg_latency: 1.28 ms
     lock_latency: 97.95 ms
        rows_sent: 732
    rows_sent_avg: 4
    rows_examined: 4245
rows_examined_avg: 22
    rows_affected: 0
rows_affected_avg: 0
       tmp_tables: 195
  tmp_disk_tables: 195
      rows_sorted: 732
sort_merge_passes: 0
           digest: 8e8c46a210908a2efc2f1e96dd998130
       first_seen: 2017-11-19 05:27:24
        last_seen: 2017-11-20 17:24:34
1 row in set (0.01 sec)
```

To know more about the `sys` schema objects, refer to https://dev.mysql.com/doc/refman/8.0/en/sys-schema-object-index.html.

Security

In this chapter, we will cover the following recipes:

- Securing installation
- Restricting networks and users
- Password-less authentication using mysql_config_editor
- Resetting the root password
- Setting up encrypted connections using X509
- Setting up SSL replication

Introduction

In this chapter, security aspects of MySQL are covered, which include restricting the network, strong passwords, using SSL, access control within a database, securing an installation, and security plugins.

Securing installation

As soon as the installation is done, it is recommended that you secure your installation using the `mysql_secure_installation` utility.

How to do it...

```
shell> mysql_secure_installation

Securing the MySQL server deployment.

Enter password for user root:
The 'validate_password' plugin is installed on the server.
The subsequent steps will run with the existing
configuration
of the plugin.
Using existing password for root.

Estimated strength of the password: 100
Change the password for root ? ((Press y|Y for Yes, any
other key for No) :

 ... skipping.
By default, a MySQL installation has an anonymous user,
allowing anyone to log into MySQL without having to have
a user account created for them. This is intended only for
testing, and to make the installation go a bit smoother.
You should remove them before moving into a production
environment.

Remove anonymous users? (Press y|Y for Yes, any other key
for No) : y
Success.

Normally, root should only be allowed to connect from
'localhost'. This ensures that someone cannot guess at
the root password from the network.

Disallow root login remotely? (Press y|Y for Yes, any other
key for No) : y
Success.

By default, MySQL comes with a database named 'test' that
anyone can access. This is also intended only for testing,
and should be removed before moving into a production
environment.
```

```
Remove test database and access to it? (Press y|Y for Yes,
any other key for No) : y
 - Dropping test database...
Success.

 - Removing privileges on test database...
Success.

Reloading the privilege tables will ensure that all changes
made so far will take effect immediately.

Reload privilege tables now? (Press y|Y for Yes, any other
key for No) : y
Success.

All done!
```

By default, the `mysqld` process runs under the `mysql` user. You can also run `mysqld` under another user by changing the ownership of all the directories used by `mysqld` (such as `datadir`, the `binlog` directory if any, tablespaces in other disks, and so on) and adding `user=<user>` in `my.cnf`. Refer to https://dev.mysql.com/doc/refman/8.0/en/changing-mysql-user.html to know more about changing the MySQL user.

It is strongly recommended not to run `mysqld` as a Unix root user. One reason is that any user with the `FILE` privilege is able to cause the server to create files as root.

The FILE privilege

Be cautious while granting the `FILE` privilege to any user because the user can write a file anywhere in the filesystem with privileges of the `mysqld` daemon, which includes the server's `data directory`. However, they cannot overwrite existing files. Also, users can read any file accessible to MySQL (or the user that runs `mysqld`) into a database table. `FILE` is a global privilege, meaning you cannot restrict it to a particular database:

```
mysql> SHOW GRANTS;
+------------------------------------------------------------
---------+
| Grants for company_admin@%
|
+------------------------------------------------------------
---------+
| GRANT FILE ON *.* TO `company_admin`@`%`
|
| GRANT SELECT, INSERT, CREATE ON `company`.* TO
`company_admin`@`%` |
+------------------------------------------------------------
---------+
2 rows in set (0.00 sec)

mysql> USE company;
Database changed
mysql> CREATE TABLE hack (ibdfile longblob);
Query OK, 0 rows affected (0.05 sec)

mysql> LOAD DATA INFILE
'/var/lib/mysql/employees/salaries.ibd' INTO TABLE hack
CHARACTER SET latin1 FIELDS TERMINATED BY '@@@@@';
Query OK, 366830 rows affected (18.98 sec)
Records: 366830  Deleted: 0  Skipped: 0  Warnings: 0

mysql> SELECT * FROM hack;
```

Notice that the company user with the `FILE` privilege is able to read data from the `employees` table.

You do not need to worry about the preceding hack as the location in which files can be read and written is limited to `/var/lib/mysql-files` by default, using the `secure_file_priv` variable. The problem arises when you set the `secure_file_priv` variable to `NULL`, an empty string, the MySQL `data directory`, or any sensitive directory that MySQL has access to (for example, the tablespaces outside the MySQL `data directory`). If you set `secure_file_priv` to a non-existent directory, it leads to an error.

It is recommended to leave `secure_file_priv` as default:

```
mysql> SHOW VARIABLES LIKE 'secure_file_priv';
+------------------+-----------------------+
| Variable_name    | Value                 |
+------------------+-----------------------+
| secure_file_priv | /var/lib/mysql-files/ |
+------------------+-----------------------+
1 row in set (0.00 sec)
```

Never give anyone access to the `mysql.user` table. To know more about the security guidelines, refer to https://dev.mysql.com/doc/refman/8.0/en/security-guidelines.html and https://dev.mysql.com/doc/refman/8.0/en/security-against-attack.html.

Restricting networks and users

Do not open your database to the whole network, meaning the port on which MySQL runs (`3306`) should not be accessed from other networks. It should be open only to the application server. You can set up a firewall using iptables or the `host.access` file to restrict access to port `3306`. If you are using MySQL on the cloud, the provider will also give a firewall.

How to do it...

To test this, you can use `telnet`:

```
shell> telnet <mysql ip> 3306
# if telnet is not installed you can install it or use nc
(netcat)
```

If telnet hangs or the connection is refused, it means that the port is closed. Please note that if you see an output like this, it means that the port is not blocked:

```
shell> telnet 35.186.158.188 3306
Trying 35.186.158.188...
Connected to 188.158.186.35.bc.googleusercontent.com.
Escape character is '^]'.
FHost '183.82.17.137' is not allowed to connect to this
MySQL serverConnection closed by foreign host.
```

It means that the port is open but MySQL is restricting the access.

When creating users, avoid giving access from anywhere (the `%` option). Restrict access to an IP range or subdomain. Also restrict the user to access only the database that is needed. For example, the `read_only` user of the `employees` database should not be able to access other databases:

```
mysql> CREATE user 'employee_read_only'@'10.10.%.%'
IDENTIFIED BY '<Str0ng_P@$$word>';
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT SELECT ON employee.* TO
'employee_read_only'@'10.10.%.%';
Query OK, 0 rows affected (0.01 sec)
```

The `employee_read_only` user will be able to access only from the `10.10.%.%` subnet and access only the `employee` database.

Password-less authentication using mysql_config_editor

Whenever you enter a password using a command-line client, you might have noticed the following warning:

```
shell> mysql -u dbadmin -p'$troNgP@$$w0rd'
mysql: [Warning] Using a password on the command line
interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1345
Server version: 8.0.3-rc-log MySQL Community Server (GPL)
~
mysql>
```

If you do not pass the password in the command line and enter when it prompts, you won't get that warning:

```
shell> mysql -u dbadmin -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1334
Server version: 8.0.3-rc-log MySQL Community Server (GPL)
~
mysql>
```

However, when you are developing some scripts over the client utilities, it is difficult to use with password prompt. One way to avoid this is by storing the password in the `.my.cnf` file in the `home` directory. The `mysql` command-line utility, by default, reads the `.my.cnf` file and does not ask for a password:

```
shell> cat $HOME/.my.cnf
[client]
user=dbadmin
```

```
password=$troNgP@$$w0rd

shell> mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1396
Server version: 8.0.3-rc-log MySQL Community Server (GPL)
~
mysql>
```

Notice that you are able to connect without giving any password, but this leads to a security concern; the password is in cleartext. To overcome this, MySQL has introduced `mysql_config_editor`, which stores the password in encrypted format. The file can be decrypted by client programs (only used in memory) to connect to the server.

How to do it...

Create the `.mylogin.cnf` file using `mysql_config_editor`:

```
shell> mysql_config_editor set --login-path=dbadmin_local --
host=localhost --user=dbadmin --password
Enter password:
```

You can add multiple hostnames and passwords by changing the login path. If the password is changed, you can run this utility again, which updates the password in the file:

```
shell> mysql_config_editor set --login-path=dbadmin_remote -
-host=35.186.157.16 --user=dbadmin --password
Enter password:
```

If you want to log in to `35.186.157.16` using the `dbadmin` user, you can simply execute `mysql --login-path=dbadmin_remote`:

```
shell> mysql --login-path=dbadmin_remote
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 215074
~
mysql> SELECT @@server_id;
+-------------+
| @@server_id |
+-------------+
|         200 |
+-------------+
1 row in set (0.00 sec)
```

To connect to `localhost`, you can simply execute `mysql` or `mysql --login-path=dbadmin_local`:

```
shell> mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1523
```

```
~
mysql> SELECT @@server_id;
+-------------+
| @@server_id |
+-------------+
|           1 |
+-------------+
1 row in set (0.00 sec)

shell> mysql --login-path=dbadmin_local
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1524
~
mysql> SELECT @@server_id;
+-------------+
| @@server_id |
+-------------+
|           1 |
+-------------+
1 row in set (0.00 sec)
```

If the password for `dbadmin` is the same across all your servers, you can connect to any of them by specifying the hostname. You do not need to specify the password:

```
shell> mysql -h 35.198.210.229
Welcome to the MySQL monitor.  Commands end with ; or \g.
~
mysql> SELECT @@server_id;
+-------------+
| @@server_id |
+-------------+
|         364 |
+-------------+
1 row in set (0.00 sec)
```

If you want to print all the login paths, do this:

```
shell> mysql_config_editor print --all
[dbadmin_local]
user = dbadmin
password = *****
host = localhost
```

```
[dbadmin_remote]
user = dbadmin
password = *****
host = 35.186.157.16
```

You can notice that the utility masks the passwords. If you try to read the file, you will only see gibberish characters:

```
shell> cat .mylogin.cnf
 ?-z???|???-B????dU?bz4-?W???g?q?BmV?????K?I?? h%?+b???_??
@V???vli?J???X`?qP
```

This utility only helps you to avoid storing cleartext passwords and ease the process of connecting to MySQL. There are many methods to decrypt the passwords stored in the `.mylogin.cnf` file. So do not think that the password is safe if you use `mysql_config_editor`. Instead of creating the `.mylogin.cnf` file every time, you can copy this file to other servers also (this works only if the username and password are the same).

Resetting the root password

If you forget the root password, you can reset it by two methods, explained as follows.

How to do it...

Let's get into the details.

Using init-file

On Unix-like systems, you stop the server and start it by specifying init-file. You can save the `ALTER USER 'root'@'localhost' IDENTIFIED BY 'New$trongPass1'` SQL code in that file. MySQL executes the contents of the file at startup, changing the password of the root user:

1. Stop the server:

```
shell> sudo systemctl stop mysqld
shell> pgrep mysqld
```

2. Save the SQL code in `/var/lib/mysql/mysql-init-password`; make it readable to MySQL only:

```
shell> vi /var/lib/mysql/mysql-init-password
ALTER USER 'root'@'localhost' IDENTIFIED BY
'New$trongPass1';

shell> sudo chmod 400 /var/lib/mysql/mysql-init-
password

shell> sudo chown mysql:mysql /var/lib/mysql/mysql-
init-password
```

3. Start the MySQL server with the `--init-file` option and other options as required:

```
shell> sudo -u mysql /usr/sbin/mysqld --daemonize --
pid-file=/var/run/mysqld/mysqld.pid --user=mysql --
init-file=/var/lib/mysql/mysql-init-password
mysqld will log errors to /var/log/mysqld.log
mysqld is running as pid 28244
```

4. Verify the error log file:

```
shell> sudo tail /var/log/mysqld.log
~
2017-11-27T07:32:25.219483Z 0 [Note] Execution of
init_file '/var/lib/mysql/mysql-init-password' started.
2017-11-27T07:32:25.219639Z 4 [Note] Event Scheduler:
scheduler thread started with id 4
2017-11-27T07:32:25.223528Z 0 [Note] Execution of
init_file '/var/lib/mysql/mysql-init-password' ended.
2017-11-27T07:32:25.223610Z 0 [Note] /usr/sbin/mysqld:
ready for connections. Version: '8.0.3-rc-log'  socket:
'/var/lib/mysql/mysql.sock'  port: 3306  MySQL
Community Server (GPL)
```

5. Verify that you are able to log in with the new password:

```
shell> mysql -u root -p'New$trongPass1'
mysql: [Warning] Using a password on the command line
interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or
\g.
Your MySQL connection id is 15
Server version: 8.0.3-rc-log MySQL Community Server
(GPL)
~
mysql>
```

6. Now, the most important thing! Remove the `/var/lib/mysql/mysql-init-password` file:

```
shell> sudo rm -rf /var/lib/mysql/mysql-init-password
```

7. Optionally, you can stop the server and start it normally without the `--init-file` option.

Using --skip-grant-tables

In this method, you stop the server and start it by specifying `--skip-grant-tables`, which will not load the grant tables. You can connect to the server as root without a password and set the password. Since the server runs without grants, it is possible for users from other networks to connect to the server. So as of MySQL 8.0.3, `--skip-grant-tables` automatically enables `--skip-networking`, which does not allow remote connections:

1. Stop the server:

   ```
   shell> sudo systemctl stop mysqld
   shell> ps aux | grep mysqld | grep -v grep
   ```

2. Start the server with the `--skip-grant-tables` option:

   ```
   shell> sudo -u mysql /usr/sbin/mysqld --daemonize --
   pid-file=/var/run/mysqld/mysqld.pid --user=mysql --
   skip-grant-tables
   mysqld will log errors to /var/log/mysqld.log
   mysqld is running as pid 28757
   ```

3. Connect to MySQL without a password, execute `FLUSH PRIVILEGES` to reload the grants, and alter the user to change the password:

   ```
   shell> mysql -u root
   Welcome to the MySQL monitor.  Commands end with ; or
   \g.
   Your MySQL connection id is 6
   Server version: 8.0.3-rc-log MySQL Community Server
   (GPL)
   ~
   mysql> FLUSH PRIVILEGES;
   Query OK, 0 rows affected (0.04 sec)
   ```

```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY
'New$trongPass1';
Query OK, 0 rows affected (0.01 sec)
```

4. Test the connection to MySQL with the new password:

```
shell> mysql -u root -p'New$trongPass1'
mysql: [Warning] Using a password on the command line
interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or
\g.
Your MySQL connection id is 7
~
mysql>
```

5. Restart the MySQL server:

```
shell> ps aux | grep mysqld | grep -v grep
mysql    28757  0.0 13.3 1151796 231724 ?      Sl
08:16   0:00 /usr/sbin/mysqld --daemonize --pid-
file=/var/run/mysqld/mysqld.pid --user=mysql --skip-
grant-tables
shell> sudo kill -9 28757
shell> ps aux | grep mysqld | grep -v grep
shell> sudo systemctl start mysqld
shell> ps aux | grep mysqld | grep -v grep
mysql    29033  5.3 16.8 1240224 292744 ?      Sl
08:27   0:00 /usr/sbin/mysqld --daemonize --pid-
file=/var/run/mysqld/mysqld.pid
```

Setting up encrypted connections using X509

If the connections between the client and MySQL server are not encrypted, anyone who has access to the network could inspect the data. If the client and server are in different data centers, it is recommended to use encrypted connections. By default, MySQL 8 uses an encrypted connection, but if the encrypted connection fails, it falls back to an unencrypted connection. You can test that by checking the status of the `Ssl_cipher` variable. If the connection is established by `localhost`, cipher won't be used:

```
mysql> SHOW STATUS LIKE 'Ssl_cipher';
+---------------+--------------------+
| Variable_name | Value              |
+---------------+--------------------+
| Ssl_cipher    | DHE-RSA-AES256-SHA |
+---------------+--------------------+
1 row in set (0.00 sec)
```

If you are not using SSL, `Ssl_cipher` will be blank.

You can mandate some users to connect only through an encrypted connection (by specifying the `REQUIRE SSL` clause) and leave it as optional for other users.

As per the MySQL documentation:

MySQL supports encrypted connections between clients and the server using the **TLS** (**Transport Layer Security**) protocol. TLS is sometimes referred to as **SSL** (**Secure Sockets Layer**) but MySQL does not actually use the SSL protocol for encrypted connections because its encryption is weak. TLS uses encryption algorithms to ensure that data received over a public network can be trusted. It has mechanisms to detect data change, loss, or replay. TLS also incorporates algorithms that provide identity verification using the X509 standard.

In this section, you will learn about setting up SSL connections using X509.

All the SSL (X509) related files (`ca.pem`, `server-cert.pem`, `server-key.pem`, `client-cert.pem`, and `client-key.pem`) are created by MySQL during installation and kept under the `data directory`. The server needs the `ca.pem`, `server-cert.pem`, and `server-key.pem` files, and the clients use the `client-cert.pem` and `client-key.pem` files to connect to the server.

How to do it...

1. Verify the files in the `data directory`, update `my.cnf`, restart the server, and check the SSL-related variables. In MySQL 8, by default, the following values are set:

```
shell> sudo ls -lhtr /var/lib/mysql | grep pem
-rw-------. 1 mysql mysql 1.7K Nov 19 13:53 ca-key.pem
-rw-r--r--. 1 mysql mysql 1.1K Nov 19 13:53 ca.pem
-rw-------. 1 mysql mysql 1.7K Nov 19 13:53 server-
key.pem
-rw-r--r--. 1 mysql mysql 1.1K Nov 19 13:53 server-
cert.pem
-rw-------. 1 mysql mysql 1.7K Nov 19 13:53 client-
key.pem
-rw-r--r--. 1 mysql mysql 1.1K Nov 19 13:53 client-
cert.pem
-rw-------. 1 mysql mysql 1.7K Nov 19 13:53
private_key.pem
-rw-r--r--. 1 mysql mysql  451 Nov 19 13:53
public_key.pem
```

```
shell> sudo vi /etc/my.cnf
[mysqld]
ssl-ca=/var/lib/mysql/ca.pem
ssl-cert=/var/lib/mysql/server-cert.pem
ssl-key=/var/lib/mysql/server-key.pem
```

```
shell> sudo systemctl restart mysqld
```

```
mysql> SHOW VARIABLES LIKE '%ssl%';
+---------------+--------------------------------+
| Variable_name | Value                          |
+---------------+--------------------------------+
| have_openssl  | YES                            |
| have_ssl      | YES                            |
| ssl_ca        | /var/lib/mysql/ca.pem          |
| ssl_capath    |                                |
| ssl_cert      | /var/lib/mysql/server-cert.pem |
| ssl_cipher    |                                |
| ssl_crl       |                                |
| ssl_crlpath   |                                  |
| ssl_key       | /var/lib/mysql/server-key.pem  |
+---------------+----------------------------------+
9 rows in set (0.01 sec)
```

2. Copy the `client-cert.pem` and `client-key.pem` files from the server's `data directory` to the client location:

```
shell> sudo scp -i $HOME/.ssh/id_rsa
/var/lib/mysql/client-key.pem /var/lib/mysql/client-
cert.pem <user>@<client_ip>:
# change the ssh private key path as needed.
```

3. Connect to the server by passing the `--ssl-cert` and `--ssl-key` options:

```
shell> mysql --ssl-cert=client-cert.pem --ssl-
key=client-key.pem -h 35.186.158.188
Welcome to the MySQL monitor.  Commands end with ; or
\g.
Your MySQL connection id is 666
Server version: 8.0.3-rc-log MySQL Community Server
(GPL)
~
mysql>
```

4. Mandate the user to connect only by X509:

```
mysql> ALTER USER `dbadmin`@`%` REQUIRE X509;
Query OK, 0 rows affected (0.08 sec)
```

5. Test the connection:

```
shell> mysql --login-path=dbadmin_remote -h
35.186.158.188 --ssl-cert=client-cert.pem --ssl-
key=client-key.pem
Welcome to the MySQL monitor.  Commands end with ; or
\g.
Your MySQL connection id is 795
Server version: 8.0.3-rc-log MySQL Community Server
(GPL)
```

```
~
mysql> ^DBye
```

6. If you do not specify `--ssl-cert` or `--ssl-key`, you will not be able to log in:

```
shell> mysql --login-path=dbadmin_remote -h
35.186.158.188
ERROR 1045 (28000): Access denied for user
'dbadmin'@'35.186.157.16' (using password: YES)

shell> mysql --login-path=dbadmin_remote -h
35.186.158.188 --ssl-cert=client-cert.pem
mysql: [ERROR] SSL error: Unable to get private key
from 'client-cert.pem'
ERROR 2026 (HY000): SSL connection error: Unable to get
private key

shell> mysql --login-path=dbadmin_remote -h
35.186.158.188 --ssl-key=client-key.pem
mysql: [ERROR] SSL error: Unable to get certificate
from 'client-key.pem'
ERROR 2026 (HY000): SSL connection error: Unable to get
certificate
```

By default, all SSL-related files are kept in the60;`data directory`. If you want to keep them elsewhere, you can set `ssl_ca`, `ssl_cert`, and `ssl_key` in the `my.cnf` file and restart the server. You can generate a new set of SSL files through either MySQL or OpenSSL. To know the more detailed steps, refer to https://dev.mysql.com/doc/refman/8.0/en/creating-ssl-rsa-files.html. There are many other authentication plugins available. You can refer to https://dev.mysql.com/doc/refman/8.0/en/authentication-plugins.html to know more details.

Setting up SSL replication

If you enable SSL replication, the binary log transfer between master and slave will be sent through an encrypted connection. This is similar to the server/client connection explained in the preceding section.

How to do it...

1. **On the master**, as explained in the preceding section, you need to enable SSL.
2. **On the master**, copy the `client*` certificates to the slave:

```
mysql> sudo scp -i $HOME/.ssh/id_rsa
/var/lib/mysql/client-key.pem /var/lib/mysql/client-
cert.pem <user>@<client_ip>:
```

3. **On the slave**, create the `mysql-ssl` directory to hold the SSL-related files and set the permissions correctly:

```
shell> sudo mkdir /etc/mysql-ssl
shell> sudo cp client-key.pem client-cert.pem
/etc/mysql-ssl/
shell> sudo chown -R mysql:mysql /etc/mysql-ssl
shell> sudo chmod 600 /etc/mysql-ssl/client-key.pem
shell> sudo chmod 644 /etc/mysql-ssl/client-cert.pem
```

4. **On the slave**, execute the `CHANGE_MASTER` command with the SSL-related changes on the slave:

```
mysql> STOP SLAVE;

mysql> CHANGE MASTER TO MASTER_SSL=1,
MASTER_SSL_CERT='/etc/mysql-ssl/client-cert.pem',
MASTER_SSL_KEY='/etc/mysql-ssl/client-key.pem';

mysql> START SLAVE;
```

5. Verify the slave's status:

```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row
***************************
               Slave_IO_State: Waiting for master to
```

```
send event
                  Master_Host: 35.186.158.188
~
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
~
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 354
              Relay_Log_Space: 949
              Until_Condition: None
               Until_Log_File: 
                Until_Log_Pos: 0
           Master_SSL_Allowed: Yes
           Master_SSL_CA_File: /etc/mysql-ssl/ca.pem
           Master_SSL_CA_Path: 
              Master_SSL_Cert: /etc/mysql-ssl/client-
cert.pem
            Master_SSL_Cipher: 
               Master_SSL_Key: /etc/mysql-ssl/client-
key.pem
        Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
~
                  Master_UUID: fe17bb86-cd30-11e7-bc3b-
42010a940003
             Master_Info_File: mysql.slave_master_info
                    SQL_Delay: 0
          SQL_Remaining_Delay: NULL
      Slave_SQL_Running_State: Slave has read all relay
log; waiting for more updates
~
1 row in set (0.00 sec)
```

6. Once you have made the SSL-related changes on all the slaves, on the master, enforce the replication user to use X509:

```
mysql> ALTER USER `repl`@`%` REQUIRE X509;
Query OK, 0 rows affected (0.00 sec)
```

Note that, this can affect other replication users. As an alternative, you can create one replication user with SSL and one normal replication user.

7. Verify the slave status on all slaves.