



C o m m u n i t y E x p e r i e n c e D i s t i l l e d

Getting Started with MariaDB

Learn how to use MariaDB to store your data easily and hassle-free

Daniel Bartholomew

[PACKT] open source*
PUBLISHING community experience distilled

Getting Started with MariaDB

Learn how to use MariaDB to store your data easily and hassle-free

Daniel Bartholomew



BIRMINGHAM - MUMBAI

Getting Started with MariaDB

Copyright © 2013 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, and its dealers and distributors will be held liable for any damages caused or alleged to be 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.

First published: October 2013

Production Reference: 1151013

Published by Packt Publishing Ltd.

Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78216-809-6

www.packtpub.com

Cover Image by Suresh Mogre (suresh.mogre.99@gmail.com)

Credits

Author

Daniel Bartholomew

Project Coordinator

Joel Goveya

Reviewers

P. R. Karthik

Daniel Parnell

Stephane Varoqui

Proofreader

Tarsonia Sanghera

Indexers

Rekha Nair

Mehreen Deshmukh

Acquisition Editor

Rubal Kaur

Production Coordinator

Adonia Jones

Commissioning Editor

Poonam Jain

Cover Work

Adonia Jones

Technical Editor

Sharvari H. Baet

About the Author

Daniel Bartholomew has been using Linux since 1997 and databases since 1998. He has written dozens of articles for various magazines, including The Linux Journal, Linux Pro, Ubuntu, User and Tux.

Daniel has been involved with the MariaDB project shortly after it began in early 2009. He currently works for SkySQL and splits his time between MariaDB documentation and maintaining the bits and pieces (including build, e-mail, web, and other servers) that keep the MariaDB project running smoothly. In addition to his day-to-day responsibilities, Daniel also serves as the MariaDB release coordinator and has been deeply involved with almost every MariaDB release.

He lives in Raleigh, North Carolina, USA with his lovely wife and four children.

I'd like to thank Amy, Ila, Lizzy, Anthon, and Rachel for their patience with me throughout the writing of this book. Thanks also to Vladislav Vaintroub, Sanja Byelkin, Roger Bartholomew, and others who were very helpful at various points during the project. Lastly, I'd like to thank Monty and the rest of the MariaDB team for the excellent database that they've created.

About the Reviewers

P. R. Karthik started his career as a MySQL DBA. Currently, he is working as a Senior MySQL DBA in Yahoo. They are managing one of the biggest MySQL farms in the world.

I would like to thank my parents and friends for their support with reviewing this book.

Daniel Parnell lives in Melbourne, Australia. He started messing around with computers when he was very young. His first computer was an AIM65 with a 4 K of RAM, BASIC and Forth in ROM, and no storage. From there, he tinkered with various home computers ranging from the Apple II to the Commodore Amiga. These days Daniel works on web and desktop apps. When Daniel is not programming or tinkering with his latest hardware project, he is spending time with his wife and two children.

Stephane Varoqui is a senior MariaDB and MySQL consultant at SkySQL. He is based in Paris. Before joining SkySQL in 2011, he worked at MySQL/Sun/Oracle as a lead MySQL consultant in the Europe/Middle East/ Africa (EMEA) region for six years.

www.PacktPub.com

Support files, eBooks, discount offers, and more

You might want to visit www.PacktPub.com for support files and downloads related to your book.

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.



<http://PacktLib.PacktPub.com>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface	1
Chapter 1: Installing MariaDB	7
Choosing a MariaDB series	8
The development series	8
The stable series	9
The maintenance series	9
Installing MariaDB on Windows	10
Installing MariaDB on Mac OS X	13
Installing MariaDB on Debian, Ubuntu, and Linux Mint	14
Installing MariaDB on Fedora, Red Hat, and CentOS	15
Installing MariaDB on other Linux distributions	17
MariaDB package security	18
After the installation	18
Troubleshooting installation issues	19
Summary	20
Chapter 2: Configuring MariaDB	21
Filesystem layout for MariaDB	21
Filesystem layout for Windows	21
Filesystem layout for Linux	22
Modular configuration on Linux	23
The MariaDB configuration file	25
Comments	25
Groups	26
Options which do not require values	26
Options which require values	27
Options, options everywhere	28
Activating configuration changes	28
Summary	29

Chapter 3: MariaDB Security	31
Securing MariaDB in ten seconds	32
Connecting safely	34
Server security	36
Building security	37
Internal network security	38
Internet security	39
Summary	39
Chapter 4: MariaDB User Account Management	41
User privileges	41
Global administrative privileges	42
Database, table, and column privileges	42
Miscellaneous privileges and limits	43
Creating users	44
Granting permissions	45
Adding and removing privileges	47
Showing grants	47
Changing passwords	48
Removing users	48
Summary	49
Chapter 5: Using MariaDB	51
Running the mysql command-line client	51
Using a database	52
Listing all databases on a server	53
Creating and dropping databases	54
Creating a database	54
Dropping a database	55
Creating, altering, and dropping tables	56
Creating a table	56
Showing the command used to create a table	58
Exploring the structure of a table	59
Altering a table	60
Adding a column	60
Modifying a column	60
Dropping a column	61
Dropping a table	61
Selecting, inserting, updating, and deleting data	61
Inserting data	62
Updating data	63
Deleting data	64

Reading data	64
Summary	68
Chapter 6: MariaDB Maintenance	69
MariaDB log files	69
The all-important binary log	69
The error log	70
The general query log	71
The slow query log	71
Optimizing and tuning MariaDB	72
Backing up MariaDB	73
Basic backups with mysqldump	73
Restoring and importing data with mysqlimport	75
Making backups of MyISAM tables quickly with mysqlhotcopy	75
Making cold backups	76
Checking and repairing tables	76
Checking and optimizing tables with mysqlcheck	76
Repairing tables	77
Summary	78
Appendix: MariaDB Next Steps	79
Index	81

Preface

Databases are all around us. Almost every website we visit and nearly every store we shop at has a database (or several) working quietly behind the scenes. The same goes for banks, hospitals, government agencies, theaters, doctors, amusement parks, and police departments. All use databases to store, sort, and analyze information.

This information comes in many forms and can be anything that can be stored electronically inside a computer. This includes books, catalogs, addresses, names, dates, finances, pictures, money, passwords, documents, preferences, tweets, posts, likes, blogs, articles, and many more. Databases are one of the primary pillars of modern life.

Your posts on Facebook and tweets on Twitter are stored in a database. All your financial information at your bank is stored in a database and so is your purchase history at your favorite online retailer. Your progress in your favorite online game? You guessed it. The record of when you last paid your water bill. You just can't get away from databases. They are quite literally everywhere.

There is a new database that has caught attention of the database community over the past few years like few others have. Its name is MariaDB; it is named after the youngest daughter of its creator, Michael "Monty" Widenius. First released in 2009, MariaDB may be relatively new, but it has a stellar parentage. It's a next-generation version of the popular MySQL database, also created by Monty. (you may have heard of it, but don't worry if you haven't).

MariaDB is *open source*. This means the source code is freely downloadable and is governed by a license that helps ensure the source code stays free and open to all. The MariaDB developers have also kindly provided installers for various operating systems.

Since its first release, MariaDB has gained a large, loyal following quicker than almost any other database. Today it powers tens of thousands of websites, big and small and is the database of choice for many companies in a wide variety of industries around the world, with hundreds of thousands of users.

The great news is that we can install and use it ourselves, right now, on our personal laptop and desktop computers. For all of its power, and make no mistake, MariaDB is a very powerful and capable database; it is very easy to install and use.

This book provides an introduction to MariaDB—enough to get us started. Don't worry if you've never used a database before, they're not that hard to understand. Before we know it we'll be on our way to becoming an expert database administrator (DBA). But even if we never move beyond just tinkering or playing around with MariaDB, we'll have learned about one of the fundamental technologies of our times.

Not a bad accomplishment over a weekend or two.

What this book covers

Chapter 1, Installing MariaDB, explains how to install MariaDB on Windows and Linux.

Chapter 2, Configuring MariaDB, explains the basics of configuring MariaDB, including the location of the configuration files, and how to set common configuration options.

Chapter 3, MariaDB Security, gives out the best practices for MariaDB security and how to easily secure a new MariaDB installation.

Chapter 4, MariaDB User Account Management, explains how to add and administer MariaDB user accounts.

Chapter 5, Using MariaDB, explains the basics of using MariaDB, including adding and dropping databases and tables, and selecting, inserting, and updating data.

Chapter 6, MariaDB Maintenance, explains how to maintain your MariaDB database and keep it running smoothly.

Appendix, MariaDB Next Steps, will provide the user with the locations of official sources of information and documentation.

What you need for this book

To get the most out of this book, we will need a computer running Windows, with any version from XP through Windows 8, a computer running MacOS X, or a computer running one of the following Linux distributions: Ubuntu, Mint, Debian, Fedora, CentOS, or Red Hat. MariaDB runs on many more operating systems and distributions, but those are the ones that are specifically mentioned and talked about in this book.

To install MariaDB we will need an Internet connection and the necessary administrative rights to install software.

To edit MariaDB configuration files, we will need a text editor. Notepad is a good universal choice on Windows. TextEdit and TextWrangler work well on MacOS X. There are many text editors on Linux, just pick a favorite: Vim, gedit, nano, emacs, and many more are all good. A word processor such as Word, Wordpad, OpenOffice, or Libre Office will not work.

No other software is required.

Who this book is for

This book is for anyone interested in learning about MariaDB or databases in general. It doesn't assume any prior database experience, however it does assume prior computer experience. You should be comfortable with installing software, editing configuration files, and using the command line or terminal.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: "All of them can be changed later by editing the `my.ini` file."

A block of code is set as follows:

```
SET PASSWORD FOR <user> = PASSWORD('<password>');
```


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


```
SET PASSWORD FOR <user> = PASSWORD('<password>');
```

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

```
sudo mv -vi MariaDB.repo /etc/yum.repos.d/
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Click on **Finish** to quit the installer."

[ Warnings or important notes appear in a box like this.]

[ Tips and tricks appear like this.]

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book – what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to feedback@packtpub.com, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books – maybe a mistake in the text or the code – we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from <http://www.packtpub.com/support>.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at questions@packtpub.com if you are having a problem with any aspect of the book, and we will do our best to address it.

1

Installing MariaDB

Before we can start using MariaDB, we have to install it. The MariaDB source code can be compiled to run on a wide variety of different platforms and system architectures, but there are pre-compiled packages available for Windows and Linux which make the process much easier.

There are several packages types, including the source code:

- Windows MSI packages
- Linux YUM packages
- Linux APT packages
- Linux and Windows binaries
- Source code

Windows MSI packages are for computers and servers running Windows 8, Windows XP, and everything in between. Linux `.rpm` packages are used on distributions, such as Fedora, CentOS, and Red Hat, which use the **Yellow Dog Updater, Modified (YUM)** package manager. Linux `.deb` packages are used on distributions, such as Debian and Ubuntu, which use the **Advanced Packaging Tool (APT)** package manager. We will cover how to install all these types in this chapter.

We will cover the fourth type, Linux and Windows Binaries, briefly. These packages are mainly useful to experienced users of MariaDB who have nonstandard setups. The Windows binaries come in a ZIP file (`.zip`) and the Linux binaries in a gzipped tar file (`.tar.gz`).

Even though MariaDB binaries are recommended for more experienced users, installing them is not especially difficult. Check the following links for the official instructions for installing the Linux and Windows binary packages, respectively:

- <https://mariadb.com/kb/en/installing-mariadb-binary-tarballs>
- <https://mariadb.com/kb/en/installing-mariadb-windows-zip-packages>

We will also go over how to install MariaDB on Mac OS X. Packages for it are not supplied by the MariaDB developers, but by a third party.

The choice of which type of package to install is an easy one, just use whichever is appropriate for your system. If you are on Windows, use the MSI package. If you are on Ubuntu or Debian, use the APT packages. And if you are on Red Hat, Fedora, or CentOS, use the YUM packages.

The next few sections contain instructions for each type, but before we get to that we need to talk about series. And no, it has nothing to do with baseball, but it does lend itself to a baseball analogy.

Choosing a MariaDB series

MariaDB development proceeds along multiple development tracks called **series**. There is a **stable series** and several **maintenance series**. Often, there is also a **development series**. This is similar to the Debian practice of having both a **stable** and **unstable** version.

The development series

The development series of MariaDB is where major new features and capabilities are introduced. Think of this like minor league baseball where the up and coming future stars are introduced and are polished and honed to perfection. At any given time, the quality of the current development release could range from **Alpha** (which has no guarantees that it will even work reliably) to **Beta** (which is feature complete but generally needs lots of bug fixing and testing) to **Release Candidate** (which is ready for general use except for some additional testing and minor bug fixing).

During the development cycle, there will generally be several alpha releases, where new features are introduced, followed by a couple beta releases where the code is refined and polished, followed by one or two RC releases where final fixes and polishing take place. The final step for any development series is when it is declared stable and moves into the major league stable series.



If the current development series release is an RC release, we may want to choose that over the current stable release. Otherwise, it is generally best to stick with whatever the current stable release is.

The stable series

For most users just starting out, whatever series is marked stable is the one to use. This is the major league series. The best and most complete version currently available. After a development series has reached a sufficient level of quality to be considered stable, it is promoted to this series and becomes the recommended version of MariaDB.

After being marked as stable, the MariaDB Foundation has a policy that major MariaDB version will be well supported with bug and security fixes and maintenance releases for a period of at least five years. This is regardless of whether it is the current stable series, or if it is one of the maintenance series. It all depends on when it first became stable.

The maintenance series

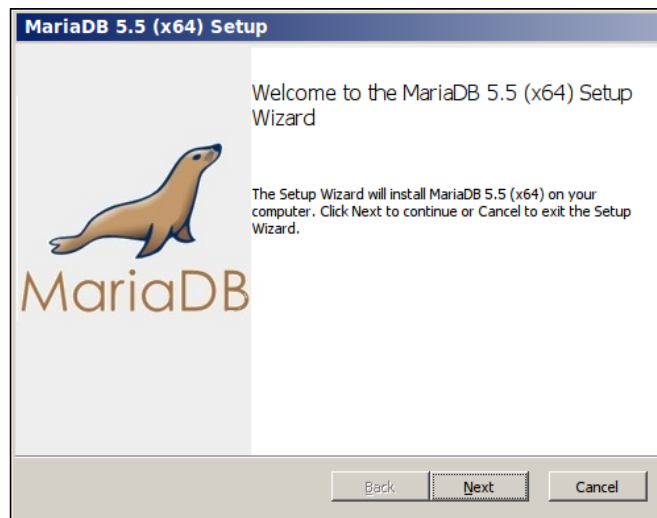
When a series moves from development to stable, whatever series was stable is moved to the maintenance series. This means that it will still receive bug fixes but it is no longer the recommended or preferred release of MariaDB. Think of it as the hall of fame – full of great previous releases of MariaDB that, while still excellent, have been replaced by a new generation. At any given time there may be three, four, or more MariaDB major versions in the maintenance series.

We'll now go through installing MariaDB for each of the major operating systems. First Windows, then Mac OS X, then Debian and Ubuntu Linux, then Fedora, Red Hat, and CentOS Linux, and lastly other Linux distributions.

Installing MariaDB on Windows

There are two types of MariaDB downloads for Windows: **ZIP files** and **MSI packages**. As mentioned previously, the ZIP files are similar to the Linux binary `.tar.gz` files and they are only recommended for experts who know they want it. If we are starting out with MariaDB on Windows, it is recommended to use the MSI packages. Here are the steps to do just that:

1. Download the MSI package from <https://downloads.mariadb.org/>. First click on the series we want (stable, most likely), then locate the Windows 64-bit or Windows 32-bit MSI package. For most computers, the 64-bit MSI package is probably the one that we want, especially if we have more than 4 Gigabytes of RAM. If you're unsure, the 32-bit package will work on both 32-bit and 64-bit computers.
2. Once the download has finished, launch the MSI installer by double-clicking on it. Depending on our settings we may be prompted to launch it automatically. The installer will walk us through installing MariaDB.



3. If we are installing MariaDB for the first time, we must be sure to set the root user password when prompted.

User settings

Default instance properties
MariaDB 5.5 (x64) database configuration

☒ **Modify password for database user 'root'**

New root password: Enter new root password

Confirm: Retype the password

☐ **Enable access from remote machines for 'root' user**

☐ **Create An Anonymous Account**

This option will create an anonymous account on this server.
Please note: this setting can lead to insecure systems.

☒ **Use UTF8 as default server's character set**

Back Next Cancel

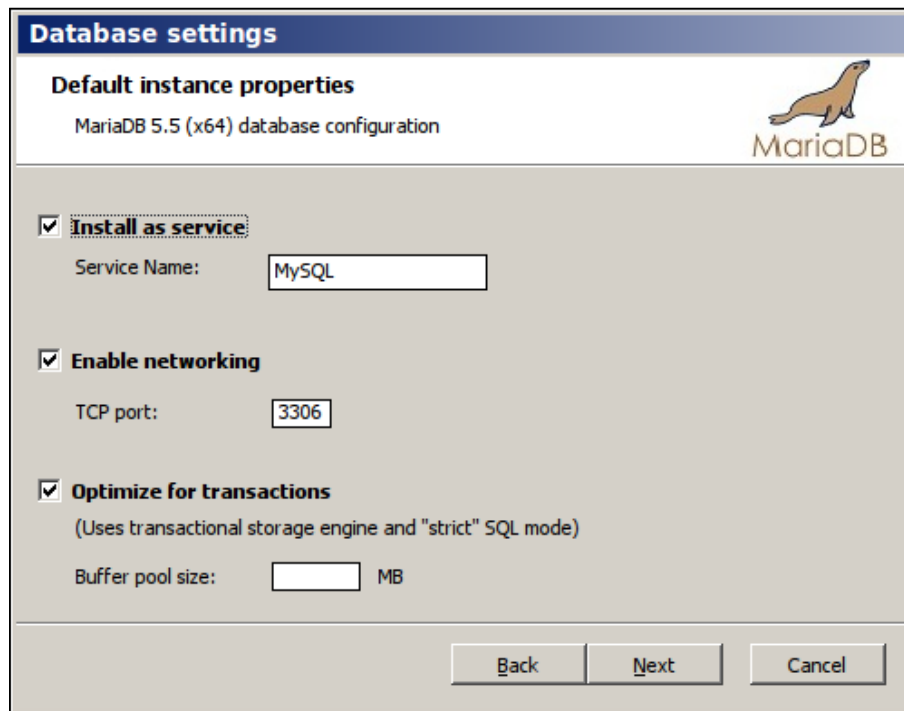
4. Unless we need to, don't enable access from remote machines for the root user or create an anonymous account. We'll cover creating regular user accounts in *Chapter 4, MariaDB User Account Management*.

☐ **Create An Anonymous Account**

This option will create an anonymous account on this server.
Please note: this setting can lead to insecure systems.

5. The **Install as service** box is checked by default, and it's recommended to keep it that way so that MariaDB starts up when the computer is booted. The **Service Name** textbox has the default value `MySQL` for compatibility reasons, but we can rename it if we like.
6. Check the **Enable networking** option, if you need to access the databases from a different computer. If we don't it's best to uncheck this box. As with the service name, there is a default TCP port number (3306) which you can change if you want to, but it is usually best to stick with the default unless there is a specific reason not to.

7. The **Optimize for transactions** checkbox is checked by default. This setting can be left as is.



8. There are other settings that we can make through the installer. All of them can be changed later by editing the `my.ini` file (more on that in *Chapter 2, Configuring MariaDB*), so we don't have to worry about setting them right away.
9. If our version of Windows has User Account Control enabled, there will be a pop-up during the installation asking if we want to allow the installer to install MariaDB. For obvious reasons, click on **Yes**.
10. After the installation completes, there will be a MariaDB folder added to the start menu. Under this will be various links, including one to the MySQL Client, which we will find out more about in *Chapter 5, Using MariaDB*.



If we already have an older version of MariaDB or MySQL running on our machine, we will be prompted to upgrade the data files for the version we are installing, it is highly recommended that we do so.

11. Eventually we will be presented with a dialog box with an installation complete message and a **Finish** button. If you got this far, congratulations! MariaDB is now installed and running on your Windows-based computer. Click on **Finish** to quit the installer.

To learn about installing MariaDB on Mac OS X or Linux, read on. Otherwise, feel free to skip to the *After the installation* section at the end of this chapter.

Installing MariaDB on Mac OS X

One of the easiest ways to install MariaDB on Mac OS X is to use **Homebrew**, which is an Open Source package manager for that platform. Before you can install it, however, you need to prepare your system. The first thing you need to do is install Xcode; Apple's integrated development environment. It's available for free in the Mac App Store.

Once Xcode is installed you can install brew. Full instructions are available on the Brew Project website at <http://mxcl.github.io/homebrew/> but the basic procedure is to open a terminal and run the following command:

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/mxcl/homebrew/go)"
```

This command downloads the installer and runs it. Once the initial installation is completed, we run the following command to make sure everything is set up properly:

```
brew doctor
```

The output of the `doctor` command will tell us of any potential issues along with suggestions for how to fix them. Once brew is working properly, you can install MariaDB with the following commands:

```
brew update
```

```
brew install mariadb
```

Unlike on Linux and Windows, brew does not automatically set up or offer to set up MariaDB to start automatically when your system boots or start MariaDB after installation. To do so, we perform the following command:

```
ln -sfv /usr/local/opt/mariadb/*.plist ~/Library/LaunchAgents
```

```
launchctl load ~/Library/LaunchAgents/homebrew.mxcl.mariadb.plist
```

To stop MariaDB, we use the unload command as follows:

```
launchctl unload ~/Library/LaunchAgents/homebrew.mxcl.mariadb.plist
```


To learn about installing MariaDB on Linux, read on. Otherwise, skip to the *After the installation* section at the end of this chapter.

Installing MariaDB on Debian, Ubuntu, and Linux Mint

The procedure for installing MariaDB on Debian, Ubuntu, and Linux Mint is easy, and starts with a visit to the Repository Configuration Tool at:

<http://downloads.mariadb.org/mariadb/repositories>

This tool is used for APT-based Linux distributions such as Debian, Ubuntu, and Mint, YUM-based Linux distributions such as Fedora, CentOS, and Red Hat, and other distributions that have support for MariaDB built-in such as Mageia, Arch Linux, and openSUSE.

Before using the tool you need to know which version of Ubuntu, Debian, or Mint to use. If you do not know, an easy way to find out is with the following command:

```
cat /etc/lsb-release
```

Type the command into the terminal and you will get an output similar to the following:

```
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=10.04
DISTRIB_CODENAME=lucid
DISTRIB_DESCRIPTION="Ubuntu 10.04.4 LTS"
```

The output shows that this machine is running Ubuntu 10.04 LTS "Lucid". So using the Repository Configuration Tool, click on **Ubuntu**, then **10.04 LTS 'lucid'**, then on the version or series of MariaDB you want to install. Lastly, click on the mirror you want to use. The tool will then output three pieces of text. The first are the commands to add the MariaDB repository to your system. The second contains the commands to install MariaDB, and the third contains the text and alternate instructions in case adding the repository using the first set of instructions did not work.

For example, the generated commands for adding a repository for MariaDB 10.0 on the 64-bit version of Ubuntu 12.04 LTS "Lucid" and using the `osuos1` mirror are as follows:

```
sudo apt-get install python-software-properties
sudo apt-key adv --recv-keys \
--keyserver keyserver.ubuntu.com 0xc9cb082a1bb943db
```

```
sudo add-apt-repository \  
'deb http://ftp.osuosl.org/pub/mariadb/repo/10.0/ubuntu precise main'
```

The first command installs the `python-software-properties` package, which contains the helper command we will use. The second command installs the GPG key that is used to sign MariaDB packages. See the *MariaDB package security* section later in this chapter for more information on this. The third command adds the repository using the `add-apt-repository` command.

The displayed installation commands are as follows:

```
sudo apt-get update  
sudo apt-get install mariadb-server
```

The `mariadb-server` package depends on the other MariaDB packages, so these two commands are all we need to install MariaDB. Once the second `apt-get` command finishes, MariaDB will be installed and running. Congratulations!

Jump ahead to the *MariaDB package security* section if you're interested in the MariaDB signing keys or skip to the *After the installation* section if you want to start using MariaDB right away.

Installing MariaDB on Fedora, Red Hat, and CentOS

The procedure for installing MariaDB on Fedora, Red Hat, and CentOS makes use of the **Yellowdog Updater, Modified (YUM)** package manager. There are two steps: first, create a repo file for MariaDB and second, install MariaDB.

To generate the required text for the repo file, we visit the MariaDB Repository Configuration Tool at: <http://downloads.mariadb.org/mariadb/repositories/>.

This tool is used for both APT-based Linux distributions such as Debian and Ubuntu, and YUM-based Linux distributions such as Fedora, CentOS, and Red Hat.

Click on the distribution we are using, the release available, and the version of MariaDB we want to install. After doing so, contents of the appropriate repo file will be displayed.

For example, the generated text for MariaDB 10.0 on the 64-bit version of CentOS 6 is:

```
# MariaDB 10.0 CentOS repository list - created 2013-03-09 20:58 UTC  
# http://mariadb.org/mariadb/repositories/  
[mariadb]
```

Installing MariaDB

```
name = MariaDB
baseurl = http://yum.mariadb.org/10.0/centos6-amd64
gpgkey=https://yum.mariadb.org/RPM-GPG-KEY-MariaDB
gpgcheck=1
```

The `gpgkey` line tells YUM where the MariaDB signing key is located. The `gpgcheck` line directs Yum to always use the signing key to verify the MariaDB packages. The first time we install MariaDB our system will not have the key so Yum will download it and install it. If Yum has never used the key before it will ask for confirmation whether it is OK to import the key. See the *MariaDB package security* section for more information on the MariaDB signing key.

Copy and paste the generated text from the repository configuration tool into a file using our favorite text editor. Naming the file something descriptive, such as `MariaDB.repo`, is recommended. Move the file to the `/etc/yum.repos.d/` folder using a command similar to the following:

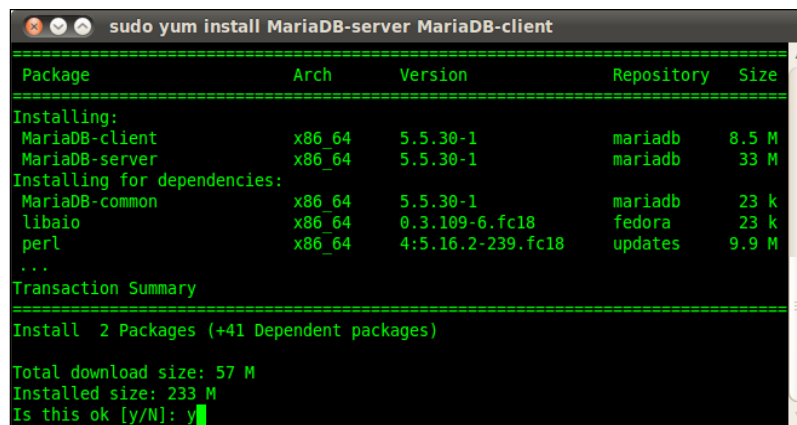
```
sudo mv -vi MariaDB.repo /etc/yum.repos.d/
```

Once the file is in place, we are ready to install MariaDB. Installing MariaDB from the command line is as simple as:

```
sudo yum install MariaDB-server MariaDB-client
```

The capitalization of the package names is important. If we type `mariadb-server` instead of `MariaDB-server`, we'll get a package cannot be found error.

YUM will gather in all of the dependencies for MariaDB and present us with a list of things we need to install. If we are installing MariaDB on a new system, the list of packages YUM installs because of dependencies could be quite large. The following screenshot shows that there are more than 41 dependent packages that will be installed when we install MariaDB.



```
sudo yum install MariaDB-server MariaDB-client
=====
Package                Arch      Version      Repository    Size
=====
Installing:
MariaDB-client          x86_64    5.5.30-1     mariadb       8.5 M
MariaDB-server          x86_64    5.5.30-1     mariadb       33 M
Installing for dependencies:
MariaDB-common          x86_64    5.5.30-1     mariadb       23 k
libaio                  x86_64    0.3.109-6.fc18 fedora        23 k
perl                   x86_64    4:5.16.2-239.fc18 updates      9.9 M
...
Transaction Summary
=====
Install 2 Packages (+41 Dependent packages)

Total download size: 57 M
Installed size: 233 M
Is this ok [y/N]: y
```

After answering *y*, the installation will get going and we will be prompted to accept the GPG signing key. We verify the fingerprint with *y*. Yum will then continue downloading and installing MariaDB and will end with a `Complete!` message.

As a final step of the installation, we start MariaDB with the following command:

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

If everything has gone well, we will see output similar to the following:

```
[dbart@fedora18-amd64 ~]$ sudo /etc/init.d/mysql start
Starting MySQL..... SUCCESS!
```

MariaDB is now installed and running. Congratulations! Jump ahead to the *After the installation* section or continue on to read about the MariaDB Package Security.

Jump ahead to the *MariaDB package security* section if you're interested in the MariaDB signing keys or skip to the *After the installation* section if you want to start using MariaDB right away.

Installing MariaDB on other Linux distributions

MariaDB is also available on several other Linux distributions and even if no formal packages are provided the MariaDB developers provide generic Linux binaries that work with many versions of Linux. Instructions on how to install and use the generic binaries are available at <https://mariadb.com/kb/en/installing-mariadb-binary-tarballs/>.

Before installing the binary packages, however, it is worth our while to look in our distribution's package manager to see if MariaDB is already there. For example, Mageia, Arch Linux, openSUSE, and others all include MariaDB in their distributions' repositories. For those Linux distributions (including these three) that the MariaDB developers are familiar with, installation instructions are provided using the MariaDB repository configuration tool (<https://downloads.mariadb.org/mariadb/repositories/>).

MariaDB package security

The packages provided by the MariaDB developers are signed with a security key so that they can be verified by package managers such as Yum and Apt. The key signing and verification infrastructure on Linux is called **Gnu Privacy Guard (GPG)**. It is a compatible Open Source version of **Pretty Good Privacy (PGP)** which is an industry standard data encryption, decryption, and verification system.

The identification number (GPG ID) of the MariaDB signing key is 0xcbbcb082a1bb943db. For long-time users of GPG, this ID may seem a little long. That's because until recently, it was common to share a short form of the GPG ID. This is discouraged now because of a GPG's vulnerability; however many utilities will still display the short form by default. The long form of the ID is more secure, so this is what the MariaDB developers share when talking about the key. But, in case we want it, the short form of the ID is 1BB943DB (it's just the last eight characters of the long form ID). For the extra cautious, the full key fingerprint is:

```
1993 69E5 404B D5FC 7D2F E43B CBCB 082A 1BB9 43DB
```

The key IDs and fingerprint are also posted in the MariaDB Knowledgebase, which is the official location of the MariaDB documentation and is available at <https://mariadb.com/kb/en/gpg/>.

By checking the signature of the packages, Linux package managers, and more importantly, we, can verify whether the package that comes from the MariaDB developers and hasn't been tampered with since they created it.

When configuring the MariaDB repository on Debian and Ubuntu, and during the initial MariaDB install on Fedora, Red Hat, and CentOS, an important task is to import the signing key. It's a good idea to verify the key by comparing it to the IDs and the fingerprint when doing so. Thankfully, this is a one-time operation. Once the key is imported the process is fully automatic. We'll only be notified if the signature check fails.

After the installation

After installing MariaDB, we can quickly test that MariaDB is up and running by opening a terminal or command-line window and running the following command (on Windows we can also open the mysql client .exe in the MariaDB folder):

```
mysql -u root -p
```

This command connects to MariaDB as the root user (-u root) and prompts for the password of that user (-p). When prompted, type in the password configured during the install. If no password was set during the install, remove -p. Until a password is set we can connect without a password.



Not having a password for the root user can be dangerous! If you did not set one during the installation, be sure to set one immediately after the install following the instructions in *Chapter 3, MariaDB Security*.

If MariaDB has been successfully installed and started, we should see something similar to the following screenshot when connecting using the previous command to launch the `mysql` command-line client:

```

dbart@fedora18-amd64
[dbart@fedora18-amd64 ~]$ sudo /etc/init.d/mysql start
Starting MySQL..... SUCCESS!
[dbart@fedora18-amd64 ~]$ mysql -u root
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 1
Server version: 5.5.30-MariaDB MariaDB Server

Copyright (c) 2000, 2013, Oracle, Monty Program Ab and others.

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

MariaDB [(none)]>

```

If you get the MariaDB command-line prompt as illustrated in the preceding screenshot, congratulations! You've just installed MariaDB and can successfully connect to the server using the command-line client. You can quit the command-line client for now. Don't worry; we'll come back to it soon.

Troubleshooting installation issues

The MariaDB installers work very well. And they are tested and retested constantly. Occasionally issues with either installing MariaDB or running it for the first time are discovered, but they are almost always fixed promptly so that users are not affected.

If we do happen to run into an issue when trying to start MariaDB, what should we do?

The first thing we should do is look in the error log. The MariaDB error log is either stored with the system logfiles (for example, under `/var/log/` on Linux) or in the MariaDB data directory. Common locations for the MariaDB data directory include `/var/lib/mysql/` on Linux, `C:\Program Files\MariaDB <version>\data\` on Windows (<version> is the version number of MariaDB we are using), and `/usr/local/var/mysql/` on Mac OS X. The error logfile itself will either be called `mysql.err` or `hostname.err` where `hostname` is the name we've given our computer. It is also worth noting that the name and location of the logfile can be customized by the `my.cnf` or `my.ini` file. The next chapter on configuring MariaDB will go into this file and its location.

Each entry inside the error logfile consists of a timestamp and a description of what went wrong at that timestamp. Sometimes the information given is enough for us to figure it out ourselves, but sometimes we may need to ask for help. We shouldn't feel bad if we can't figure an error out, even experts are sometimes stumped! If we do need to ask for help, the resources listed on the following page, especially the Maria Discuss mailing list and the official IRC channel can help greatly: <https://mariadb.com/kb/en/where-are-other-users-and-developers-of-mariadb/>

Summary

In this chapter we installed MariaDB. Our next task is to configure it, which just so happens to be the subject, and title, of the next chapter.

2

Configuring MariaDB

MariaDB is installed with a generic configuration suitable for light, casual use. This is perfect for giving MariaDB a try but hardly suitable for a production database application, or even a moderately busy website database. There are thousands of ways to tweak the settings to get MariaDB to perform the way we need it to. Many books have been written on this subject. In this chapter, we'll just cover the basics that are enough to comfortably edit the MariaDB config files and get to know our way around; think of it as a highlights tour.

Filesystem layout for MariaDB

A MariaDB installation is not a single file or even a single directory, so the first stop on our tour is a high-level overview of the filesystem layout. We'll start with Windows, and then move on to Linux.

Filesystem layout for Windows

On Windows, MariaDB is installed under a directory named with the following pattern:

```
C:\Program Files\MariaDB <major>.<minor>\
```

The following is the location for MariaDB 10.0:

```
C:\Program Files\MariaDB 10.0\
```

The only alteration of this location, unless we change it during the installation, is when the 32-bit version of MariaDB is installed on a 64-bit version of Windows. In that case, the default MariaDB directory is at the following location:

```
C:\Program Files x86\MariaDB <major>.<minor>\
```


Under the MariaDB directory on Windows, there are four primary directories: `bin`, `data`, `lib`, and `include`. There are also several configuration examples and other files under the MariaDB directory and a couple of additional directories (`docs` and `share`), but we won't go into them here. The `bin` directory is where the `.exe` files of MariaDB are located. The `data` directory is where databases are stored; it is also where the primary MariaDB configuration file, `my.ini`, is stored. We'll talk about this file later in the *The MariaDB configuration file* section. The `lib` directory contains various library and plugin files. Lastly, the `include` directory contains files useful for application developers.

We don't need to worry about the `bin`, `lib`, and `include` directories, just be aware that they exist and know what they contain. The `data` directory is where we'll spend most of the time in this chapter.

Feel free to read the next section which explains the location of MariaDB files on Linux systems, or jump ahead to the *The MariaDB configuration file* section.

Filesystem layout for Linux

On Linux distributions MariaDB follows the default filesystem layout available for Linux. For example, MariaDB binaries are placed under `/usr/bin/`, libraries are placed under `/usr/lib/`, manual pages are placed under `/usr/share/man/`, and so on. However, there are some key MariaDB-specific directories and file locations everyone should know about. Two of them have locations that are the same for the systems based on **Advanced Package Tool (APT)** and **Yellow Dog Updater, Modified (YUM)**. These locations are `/usr/share/mysql/` and `/var/lib/mysql/` respectively.

The `/usr/share/mysql/` directory contains helper scripts that are used during the initial installation of MariaDB, translations (so we can have error and system messages in different languages), and character set information. The `/var/lib/mysql/` directory is the default location for our actual database data. We don't need to worry about the contents of `/usr/share/mysql/`. It's enough to know that it exists and contains important files. Also, there is not much need to worry about the `/var/lib/mysql/` directory. MariaDB will handle the contents of that directory automatically; just know that it exists.

The next directory is where MariaDB plugins are stored. On Debian and Ubuntu systems the directory is at the following location:

```
/usr/lib/mysql/plugin/
```

As with `/usr/share/mysql/`, we don't need to worry about the contents of `/usr/lib/mysql/`. It's enough to know that it exists and contains important files. Also, if in the future we install a new MariaDB plugin, this directory is where it will go.

On YUM distributions such as Fedora, Red Hat, and CentOS, the location of the plugin directory varies depending on whether our system is 32 bit or 64 bit. If unsure, just look in both. The possible locations are as follows:

```
/lib64/mysql/plugin/  
/lib/mysql/plugin/
```

A basic rule of thumb is that if we don't have a `/lib64/` directory, we have the 32-bit version of Fedora, Red Hat, or CentOS installed.

The last directory we want to know is only found on APT-based distributions (Debian and Ubuntu) by default. It is as follows:

```
/etc/mysql/
```

The `/etc/mysql/` directory is where configuration information for MariaDB is stored. Specifically in the following two locations:

```
/etc/mysql/my.cnf  
/etc/mysql/conf.d/
```

There are some other files in the directory, but we can ignore them for now.

We'll look into more detail of the `my.cnf` file in the *The MariaDB configuration file* section. And we'll talk about the `conf.d` directory in the *Modular configuration on Linux* section.

Fedora, Red Hat, and CentOS systems don't have a `/etc/mysql/` directory by default, but they do have the `my.cnf` file and a `my.cnf.d` directory. They are at the following two location:

```
/etc/my.cnf  
/etc/my.cnf.d/
```

The `my.cnf` files function the same on all Linux versions and on Windows, where it is often named `my.ini`. The `my.cnf.d` and `conf.d` directories, despite their different names, serve the same purpose. We'll spend the rest of our highlights tour talking about the `my.cnf` file after going over the `my.cnf.d` and `conf.d` directories. Feel free to skip the next section if you are working with Windows.

Modular configuration on Linux

The `my.cnf.d` and `conf.d` directories are special locations for MariaDB configuration files. They are found on MariaDB releases for Linux such as Debian, Ubuntu, Fedora, Red Hat, and CentOS. By default, they are at the following two locations:

```
/etc/my.cnf.d/  
/etc/mysql/conf.d/
```

We will only have one of them and regardless of which one we have, they function the same. The basic idea is to allow the package manager (APT or YUM) to be able to install packages for MariaDB, which include additions to the MariaDB configuration file without needing to edit or change the main `my.cnf` file. It's easy to imagine the harm that would be caused if we installed a new plugin package and it overwrote a carefully crafted and tuned configuration file. With these special directories, the package manager can simply add a file to the appropriate directory and be done.

The MariaDB server, or any client or utility included with MariaDB, will read both the main `my.cnf` file and any files in the `my.cnf.d` or `conf.d` directories that have the extension `.cnf` when it starts.

For example, MariaDB includes a plugin called `feedback` whose sole purpose is to send back anonymous statistical information to the MariaDB developers. They use the information to help guide future development efforts. It is disabled by default but can easily be enabled by adding `feedback=on` to the `[mysqld]` group of the MariaDB configuration file (we'll talk about configuration groups in the following section). Instead of adding this to the main configuration file, we can instead create a file called `feedback.cnf` with the following contents:

```
[mysqld]
feedback=on
```

Place this file in the `my.cnf.d` or `conf.d` directory and when we start or restart the server the `feedback.cnf` file will be read and the plugin will be turned on. Doing this for a single plugin on a solitary MariaDB server may seem like too much work, but suppose we have 100 servers, and further assume that since the servers are different, each of them has a slightly different `my.cnf` configuration file. Without using these directories to turn on the feedback plugin on all of them, we would have to connect to each server in turn and manually add `feedback=on` to the `[mysqld]` group of the file. This would get tiresome and there is also a chance that we might make a mistake with one, or several of the files we edit, even if we try to automate the editing in some way. Copying a single file to each server that only does one thing (turn on the feedback plugin in our example) is much faster, and much safer. And, if we have an automated deployment system in place, copying the file to every server can be almost instant.

One thing to be aware of is that since the configuration settings in the `my.cnf.d` or `conf.d` directories are loaded after the settings in the `my.cnf` file, they override the settings in the main `my.cnf` file. This can be a good thing if that is what we want and expect. Conversely, it can be a bad thing if we are not expecting that behavior. So keep it in mind.

The MariaDB configuration file

On Windows, the MariaDB configuration file is named `my.ini` by default and is found in the data directory (see the *Filesystem layout for Windows* section to find out the data directory's location). The file can also be named `my.cnf` just as it is on Linux, and MariaDB will also look in the following additional locations for it:

```
C:\WINDOWS\my.ini
C:\WINDOWS\my.cnf
C:\my.ini
C:\my.cnf
```

On Linux, the MariaDB configuration file is always named `my.cnf` and is almost always found at one of the following two locations:

```
/etc/my.cnf
/etc/mysql/my.cnf
```

MariaDB will look for the file at both locations, but if both files exist, the options in the file MariaDB reads last will override the options it read in the first file. So to avoid confusion, we should only have one or the other and if we discover we have both for some reason, we should combine them into one file.

The configuration file is just a text file and we can edit it with our favorite text editor. Even though the extensions may be different (`.ini` or `.cnf`) the contents of the files are laid out the same. Apart from empty lines, which are ignored, there are four main types of lines in a MariaDB configuration file. These are: comments, groups, options with no values, and options with values.

Comments

Comment lines are lines that begin with `#` or `;`. Comments are ignored by MariaDB. They often contain useful information and are a great place to keep notes when we make changes to the file. Comments can also start in the middle of the line. Just think of anything from the initial comment character to the end of a line as a comment. Here are some examples:

```
# Here is a comment
; This is also a comment
port = 3306 # Here's a comment about the port number
```

Groups

Groups are sections in the configuration file. Each MariaDB program or utility can have its own configuration information in it. Even individual series of MariaDB have their own group (these are useful if we are testing a development version and want to enable a new feature without affecting older servers that use the same configuration file).

A group begins with the name of the group, enclosed in brackets ([]) on a line by itself. The group continues to the end of the file or to the beginning of the next group. Here is an example:

```
[mysqld]
# Configuration options for the mysqld program go here
```

In addition to [mysqld], other common groups include:

```
[mysql]
# configuration options for the mysql command-line client
[client-server]
# configuration options for both clients and the server
[mysqladmin]
# configuration options for the mysqladmin program
[mysqlcheck]
# configuration options for the mysqlcheck utility
[mariadb-10.0]
# configuration options just for MariaDB 10.0 series servers
```

There are many other possible groups, but I think we get the picture. We just use the ones we want and can ignore the others.

Options which do not require values

Configuration options either take a value or not. Those that do not need a value appear on a line by themselves with no equals sign (=). They are used for options that are either on or off so there is no need for arguments. If it exists in the configuration file (and isn't commented out) the feature is on. If it doesn't exist (or it is commented out) the feature is set to whatever the default is (ON or OFF). An example would be:

```
no-auto-rehash
```

To turn OFF a feature that is ON by default, just add =OFF to it as follows:

```
no-auto-rehash=OFF
```

We can also be more explicit about turning a feature on by appending =ON to an option. It's not necessary though.

One other thing worth noting is that option names are not case sensitive. We can also choose to use dashes (-) or underscores (_) in the names. For example, the following two options are the same:

- `max_allowed_packet = 1M`
- `MAX-Allowed-Packet = 1M`

One exception to this is with options that have values (described in the Options which require values section). If the value is a file or location on a case-sensitive filesystem like those used on Linux, that value will be case sensitive. The option name is not case sensitive, but the value is. For example, the first two of the following three examples work the same but the third does not (and on Linux it will almost assuredly not work):

```
socket = /var/run/mysqld/mysqld.sock
Socket = /var/run/mysqld/mysqld.sock
socket = /VAR/run/mysqld/mysqld.sock
```

To keep our `my.cnf` file readable it is best to keep option names lowercase, even though MariaDB will accept upper or mixed case. It is advised to keep options in lowercase, unless they need to be otherwise.

Options which require values

As mentioned in the previous section, some configuration options require a value of some sort to be set. For example, the default `[client]` section in the Ubuntu version of the MariaDB `my.cnf` file contains the following two options:

```
port = 3306
socket = /var/run/mysqld/mysqld.sock
```

Setting options such as `port` or `socket`, or any other setting which requires a value, without giving a value, will cause an error and MariaDB may refuse to start.

There is also a special line at the end of Linux `my.cnf` files. It begins with an exclamation point (!) and its purpose is to include the special `conf.d` or `my.cnf.d` directories. Don't change or remove this line!



Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com> and register to have the files e-mailed directly to you.

Options, options everywhere

Each individual program and utility included with MariaDB has its own set of configuration options. Run one from the command line with `--help` and we'll get a list of all the options the program has and what they are currently set to. Run the command with `--print-defaults` and we'll see the values that we've set. For example, here's the output of the `mysql` command-line client program on my local machine:

```
shell> mysql --print-defaults
mysql would have been started with the following arguments:
--port=3306 --socket=/var/run/mysqld/mysqld.sock
```

If we want to see all of the default values for a command (what they would be if we didn't have a config file), use `--no-defaults --help --verbose`. For example:

```
shell> mysqld --no-defaults --help --verbose
```

The list that gets printed by this command is quite long, so we won't show it here. And it shows more information than just the default values of options. What we're interested in for now is a table towards the end of the output that begins with:

```
Variables (--variable-name=value)
and boolean options {FALSE|TRUE}
```

Putting all of the above information into practice, I've created a fairly generic and heavily commented example `my.cnf` file. It is available in the code bundle given away with this book.

There isn't space here to go into detail on the many options available for configuring MariaDB. If we want to learn more, a good place to start is in the *Optimization and Tuning* section of the MariaDB Knowledge Base available at <https://mariadb.com/kb/en/optimization-and-tuning/>.

Activating configuration changes

The last stop on our highlights tour of MariaDB configuration is how to activate changes once we've made them. To do so, we need to reload or restart MariaDB.

On Windows, we perform the following command:

```
sc stop mysql
sc start mysql
```

These two commands assume that we set the service name to `mysql` (the default) during installation. If we set it to a different name, we would specify that instead.

On Linux systems, we generally do the following (and we may need to preface it with `sudo`):

```
/etc/init.d/mysql reload
```

Example `my.cnf` file



There's an awful lot of information in this chapter about file locations, comments, options, groups, and so on. If you're anything like me, your head is probably swimming, wondering how you're ever going to make sense of it all. To see a big picture view of how everything works together I've created an example `my.cnf` file with lots of comments to explain the different parts and settings of a typical `my.cnf` file. You can download it from the book's website.

Summary

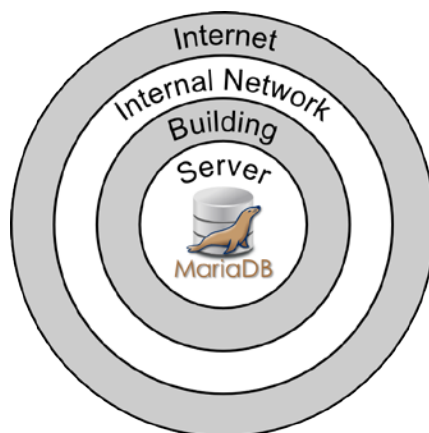
That's it for our configuration highlights tour! The next chapter is on securing MariaDB. After all, now that we know how to configure MariaDB to our liking, we wouldn't want some nefarious character to mess things up, now would we? We'll cover an easy way to secure our new installation of MariaDB, and go over basic things we can do to keep our database secure.

3

MariaDB Security

Bad things happen – accidentally and on purpose – and we want to protect our MariaDB database against both. Threats come in many different forms and come from many different places including – but not limited to – physical threats, filesystem threats, network threats, and user threats.

You can think of the data in your database as being at the center of a set of rings as illustrated by the following diagram:



The outermost ring is the **Internet**. This is the outside world. If we are running a business this is where our customers are. It's also where many attacks originate.

The next two rings are our **Internal Network** and the **Building** where our MariaDB database server is located. Internal corporate networks can span several buildings, but if we're a small business or a hobbyist the network might just be a single building or even a room or two inside a building or house. We need to be careful of the security on our internal network as we are on the external Internet, especially since more attacks come from inside networks than from the outside world.

Physical security is also important. If an attacker can simply walk in and take the server or computer our MariaDB database is present and walk out with it, none of our network and other security measures will mean anything. It's trivial for an attacker to gain access to our data if they have physical access to the machine.

The next ring is the **Server** on which MariaDB is running. Who can log in? Where can they log in from? Who has administrative rights? Does it have monitoring and backup systems in place so we can keep an eye on it? The answers to these questions depend on factors that are out of the scope of this book. We should find out the answers though.

For example, if you know that only three other people have login access to the server, we could tighten up security to a level — such as requiring SSH keys to login — that might be unacceptable on a server that has hundreds of users or is shared with other departments. Knowing who are the administrators of the server is useful because we want to know who to call if something goes wrong. It's the same for backup and monitoring systems; we need to know where they are and how to access them because if we don't they won't be of much use to us when a problem occurs.

Now we're inside the server and have come to the center of the rings — the MariaDB database itself. If it looks like MariaDB is sitting right at the center of a bullseye, that's because it is. Security starts here and there's no better time to secure our MariaDB installation than right now. We'll begin by working our way from the inside out.

Securing MariaDB in ten seconds

The first thing we need to do after we install MariaDB is to run the `mysql_secure_installation` script. This useful script ships with MariaDB and its sole purpose is to quickly and easily set up some basic security. To run it, open a command line and enter:

```
mysql_secure_installation
```

The script will ask several questions. For nearly all of them, it's best to answer yes (y). The only question we might want to answer no to is when the script asks us to set a root user password. If we've already set a root password we can safely skip this question (the script is helpful enough to tell us when it is safe).

The other questions include removing the `test` database, removing the default anonymous user, and disallowing remote root user logins. The anonymous user and test database are included in the default MariaDB installation for testing purposes, but there's almost never a reason to keep them. We can always create a new test user and database, or several, for our testing needs.

Following is the output of a complete run of the script on a server running Fedora 18:

```
[dbart@fedora18-amd64 ~]$ mysql_secure_installation
```

```
NOTE: RUNNING ALL PARTS OF THIS SCRIPT IS RECOMMENDED FOR ALL MariaDB
SERVERS IN PRODUCTION USE! PLEASE READ EACH STEP CAREFULLY!
```

```
In order to log into MariaDB to secure it, we'll need the current
password for the root user. If you've just installed MariaDB, and
you haven't set the root password yet, the password will be blank,
so you should just press enter here.
```

```
Enter current password for root (enter for none):
```

```
OK, successfully used password, moving on...
```

```
Setting the root password ensures that nobody can log into the MariaDB
root user without the proper authorization.
```

```
Set root password? [Y/n] y
```

```
New password:
```

```
Re-enter new password:
```

```
Password updated successfully!
```

```
Reloading privilege tables..
```

```
... Success!
```

```
By default, a MariaDB installation has an anonymous user, allowing anyone
to log into MariaDB 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? [Y/n] 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? [Y/n] y
... Success!
```

By default, MariaDB 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? [Y/n] 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? [Y/n] y
... Success!
```

Cleaning up...

All done! If you've completed all of the above steps, your MariaDB installation should now be secure.

As the output of the script says, after running it, our MariaDB installation is now secure. In fact, if we run it immediately after installing MariaDB the only user that will now be able to connect is `root`, and it will only be able to do so while we are logged in to or sitting in front of the actual computer MariaDB is running on. This isn't very convenient and we don't want to give other users or applications the root user password, so we'll eventually have to add users and open things up at least a little. *Chapter 4, MariaDB User Account Management*, goes into this subject.

Connecting safely

Now that the root user has a password, it's up to us to make sure the password – and the passwords of all other users we will inevitably create – stay secure. One of the most important ways to do that is to always follow a good connection practice.

When connecting to MariaDB as root, or any other user, we tell the `mysql` command line client that we are connecting with a password by using the `-p` flag. When we do so we can either specify the password right after the `-p` flag with no space between, as shown in the following command line:

```
mysql -u root -pmypassword
```

Or we can just leave the `-p` flag by itself and the client will prompt us for the password, as shown in the following command line:

```
mysql -u root -p
Enter password:
```

It is almost never a good idea to type our password on the command line. The reason is because status and system logs may record the command. This is very useful for determining who is connecting and when, but it can be very dangerous as it exposes the password to anyone who can access the logs. By using just `-p` and then entering the password when prompted, the password is not echoed to the screen and is not logged or displayed.

A situation might arise if we want to create a script that connects to our MariaDB database at certain times to do some housekeeping or other tasks. Naturally, we want the user to have rights to only do the things that it needs to do and we want the user to have a good password. Using the password prompt method will work if we use a tool such as Expect on Linux, but that may not be available, or work, in all cases. So how do we connect without exposing the password? The answer is option files.

Option files are just text files, and technically we can create one anywhere, but it should probably be in a logical location, like in the same folder that the script is in or in a hidden directory in our home directory.

The contents of the option file can be any of the options we can put into a `my.cnf` file, but for the preceding example of supplying a script with a username and password the contents are very simple, only three lines, the first starting a client section and the other two specifying the username and password to use (`scriptuser` and `scriptpassword` in this example):

```
[client]

user = scriptuser

password=scriptpassword
```

Notice that the preceding example uses spaces on the user line, but not on the password line. This is because passwords can have spaces in them, so the MariaDB `mysql` command-line client starts reading the password immediately after the equals sign. So unless the first character of our password is a space, we start the password immediately after the equals sign.

We tell the client to read the file by using the `--defaults-file` option, as follows:

```
mysql --defaults-file=/path/to/my-file
```

With the preceding in place, the client will read the file as it connects and use the username and password we supplied (along with any other client options that we add in the file).

To be safe when using this method, we should set the file so that it is readable only by the user that will run the script. We can consult our operating system documentation for the specifics on how to do this. On Linux-based systems a good command to use is:

```
chmod 600 my-file
```

The preceding command sets the file as readable and writable by the user that owns the file (6) and no access for everyone else (the two zeroes). Consult the `chmod` documentation for full details.

On Windows, we can accomplish the same thing by right-clicking on the file in the file manager, selecting **Properties** and then adjusting the access permissions. Consult the Windows documentation for full instructions.

Server security

With MariaDB itself locked down nice and tight, and with us using good password practices, we now need to look at the computer MariaDB is running on.

If we are running MariaDB on our own desktop or laptop, and we are the only one who can log in to it, then there's not much to worry about apart from the normal things we do to keep our computer secure—virus and malware protection, system updates, keeping it in a secure location, and so on. It is also useful to encrypt our hard drives, or at least our home folders using an operation supported by most modern operating systems.

When we install MariaDB on a dedicated server then there is more that we have to worry about. Servers are almost always multiuser, so as part of server security, we need to know who can log in and most importantly, who has root or administrator access. If we are the administrator of the machine, we can ensure that only those we want to have access to the administrator or root have access. If we're using MariaDB on a machine that our I. T. department gave us access to then we need to find out who has access and what their rights are, if for no other reason so that we know who has sufficient rights on the server to make changes that could be harmful.

Building security

We come to building security by continuing out to the next ring. All the protection inside the server won't do us any good if the server decides to take a walk at three in the morning. Just as we secure the inside of the server, we need to secure the outside too.

Firstly, where is the server located? Is it in a common area where anyone in the office could get to it? This could be bad on a number of levels, the first being that someone could accidentally or on purpose cut power to it. We can mitigate external power outages to some extent by installing battery backup units and such, but someone with physical access to the machine can easily get around that and cut power to our servers. To its credit, MariaDB—when we use a transactional or crash safe storage engine—guards against losing or corrupting data in such cases, but at the very least, a surprise power outage will disrupt every application that needs to talk to that database server. If the server is in a locked room, we should find out who has access to the room.

Also consider the building. Most businesses and offices close at night—the building or office is locked at closing time and opens again in the morning—however this is not true for all businesses. For example, what if the server is located in the manager's office of a 24x7 supermarket and the door to that office is always open or unlocked? If so then we need to think about locking that door (automatically if people keep forgetting to lock it), or getting a small lockable server cage installed which is bolted to the wall or floor, or come up with some other way of securing the server.

An easy analogy is to treat a server like money. We use database servers to either save money, generate income, or both, so the analogy is apt. If we would feel comfortable leaving a large stack of money in the location our server is in, then it is probably a pretty good place for our server (assuming there is power and adequate cooling).

The best place for a server is usually with other servers in a dedicated *server room*. Preferably it should be a room that is secure and where access is controlled with well-defined security policies and procedures. These could range from a locked closet (that only a few chosen people can access and which has the server sitting on a shelf) to a locked server cage at a large data center (that has a raised floor cooling, 24x7 on-site security, and everything in surplus). There is no one particular location that is right for every situation, but we need to evaluate ours and make sure our server is physically protected.

Internal network security

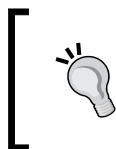
The security of the internal network is related to building security. If our MariaDB server is located in a locked server closet, then we will likely be accessing it remotely from our desk. If so, then we need to at least be aware of the security of our internal network. Some key questions to ask our local network administrator include:

- Is there a firewall in place to prevent outside access to our network?
 - If there is, great! If not, suggest that one be added.
- Is there a Wi-Fi network that is directly connected to our internal network, or is the Wi-Fi sectioned off into its own network?
 - If the Wi-Fi network is connected directly to the internal network, see if that can be changed.
- What type of access, if any, do telecommuting employees have – VPN, SSH, something else?
 - If telecommuting employees are forced into using VPN or SSH to connect, that is good, as both of those access methods are encrypted. If the answer is something else, we need to find out if it is secure and encrypted (if it isn't, we need to complain).
 - Are our database users defined with % for the network part or are they all restricted to localhost or known valid locations and networks? The % character is the wildcard character and its presence in the network part of a username means that the user named can connect from *anywhere*, which may be convenient, but is not good from a security standpoint.
- If we are in a large company, do different departments have their own segregated networks, and if so do they have access to the network the server is on?

- If our database is part of a project inside the company for a product in the early stages of development, we might not want the salespeople, for example, finding out about it until it is ready.
- At the very least, when we connect to the server remotely, we must always do so securely using SSH or an encrypted tunnel. And if we don't know how to do so we need to learn right away.

Internet security

The last ring is the outside world, that is, the Internet. Generally speaking, we don't want to expose our MariaDB database server directly to the Internet ever. It's not that MariaDB is especially vulnerable, any more than any other piece of software, it's just that it's almost never necessary to expose it to the Internet and part of good security is to not expose something unless we have to (in the same way that a poker player doesn't want to reveal his hand to the other players, or the rest of the world). When MariaDB is running on a web server, the web server software can directly connect with no need for a network connection. If our MariaDB server is separate from our web server, then we can almost always connect the two of them together over our internal network and if not, we can set up some sort of secure tunnel between the two.



If you do think you've found a legitimate reason to expose your MariaDB server to the entire Internet, I strongly encourage you to talk with one of the many fine MariaDB consulting companies and have them help you work out an alternative solution.

Summary

In this chapter, we learned a bit about how to secure our MariaDB server. Security is a big topic, and cannot possibly be covered completely in a single chapter. There are many resources, both online and offline, to help you learn more about this important topic. But don't limit yourself to books or articles about securing MariaDB or other databases; also take the time to learn about system, network, and physical security.

That said, the most secure safe in the world is one with no doors, windows, or other openings of any kind, but it's not a very useful or safe if you can't access it when you need to. So in the next chapter, *MariaDB User Account Management*, we'll make our currently secure MariaDB server a bit more useful by adding user accounts and learning how to manage them.

4

MariaDB User Account Management

The root user in our MariaDB database has rights to every database and table, we don't want to use it for day-to-day operations or hand out the login ID and password of root to anyone who doesn't absolutely need to have it. Instead, we want to create users that have specific rights to the specific databases they need to work with.

User privileges

The privileges or rights that we can grant to users are many and varied. They break down into three main categories:

- Global administrative privileges
- Database, table, and column privileges
- Miscellaneous privileges

Global administrative privileges

The following table lists the global administrative privileges. Global privileges apply to all databases and tables within those databases, which belong to the entire MariaDB database server.

Privilege	Description
CREATE USER	The ability to create a user using the CREATE USER statement.
FILE	The ability to use the LOAD DATA INFILE statement and the LOAD_FILE () function.
PROCESS	The ability to use the SHOW PROCESSLIST command.
RELOAD	The ability to use the FLUSH statement.
REPLICATION CLIENT	The ability to use the SHOW MASTER STATUS and SHOW SLAVE STATUS commands.
REPLICATION SLAVE	The ability to get updates made on the replication master server.
SHOW DATABASES	The ability to list all of the databases on the server.
SHUTDOWN	The ability to shut down the server using the mysqladmin shutdown command.
SUPER	The ability to use superuser statements such as CHANGE MASTER TO . . . , PURGE LOGS; to SET global variables; and to KILL other users' threads.

Database, table, and column privileges

The following table lists the database and table privileges. These privileges only apply to a specific database or table within a database.

Privilege	Description
ALTER	The ability to change indexes and tables.
ALTER ROUTINE	The ability to change or delete procedures and stored functions.
CREATE	The ability to create databases and tables.
CREATE ROUTINE	The ability to create procedures and stored functions.
CREATE TEMPORARY TABLES	The ability to create temporary tables.

Privilege	Description
CREATE VIEW	The ability to create views.
DELETE	The ability to delete rows from tables.
DROP	The ability to delete entire databases and tables.
EVENT	The ability to alter, create, and drop events from the event scheduler.
EXECUTE	The ability to execute stored functions and procedures.
INDEX	The ability to create or delete indexes.
INSERT	The ability to insert new rows of data into a table.
LOCK TABLES	The ability to lock and unlock tables.
SELECT	The ability to read data from a table.
SHOW VIEW	The ability to use the SHOW CREATE VIEW statement.
TRIGGER	The ability to use the CREATE TRIGGER and DROP TRIGGER statements.
UPDATE	The ability to modify rows in a table.

Column privileges apply to individual columns within a table. There are only three of them: INSERT, UPDATE, and SELECT.

Miscellaneous privileges and limits

The following table lists miscellaneous privileges which don't quite fit into either of the two previous categories.

Privilege	Description
USAGE	Grants nothing real, but can be used to change global options for a user.
ALL PRIVILEGES	Can be used to grant all available privileges to a user. Does not grant the GRANT OPTION privilege. Can be shortened to ALL.
GRANT OPTION	Gives a user the ability to give other users the privileges they have. This is given at the end of the GRANT statement. See the <i>Granting Permissions</i> section of this chapter for some examples.

There are also several limits we can place on user accounts. These are given in the following:

Limit	Description
MAX_QUERIES_PER_HOUR	The number of SQL statements or queries the user account can issue per hour. This includes updates.
MAX_UPDATES_PER_HOUR	The number of SQL update statements (not queries) the user account can issue per hour.
MAX_CONNECTIONS_PER_HOUR	The number of connections the user account can start per hour.
MAX_USER_CONNECTIONS	The number of simultaneous connections to the database server the user account can have. If set to zero, the number will be equal to the <code>max_connections</code> setting. If the <code>max_connections</code> setting is also zero then there is no limit to the number of simultaneous connections.

Full documentation of the various privileges can be found at <https://mariadb.com/kb/en/grant/>.

Creating users

Creating a user in MariaDB is a two-step process. First, we create the user using the `CREATE USER` statement, and then we give, or `GRANT`, the user the privileges we want them to have. We'll go over the `CREATE USER` statement in this section and the `GRANT` statement in the *Granting permissions* section.

A `CREATE USER` statement has the following pattern:

```
CREATE USER 'username'@'host' IDENTIFIED BY 'password';
```

We customize the username, host, and password parts to the appropriate values. If we don't want to specify a password (not recommended!) then we can drop the `IDENTIFIED BY 'password'` part. This and all SQL statements that we input into MariaDB need to end with a semicolon (;).

The host part can be several things. It can be the hostname of the computer the user connects from, the IP address of the computer the user connects from, the network the user connects from, or it can be the wildcard symbol `%`, which means any host.

Here are some examples. This first example user can login from anywhere because of the wildcard character, %, in the host part. The user's password is: bomber.

```
CREATE USER 'boyd'@ '%' IDENTIFIED BY 'bomber';
```

The following three examples demonstrate using various host names. The first specifies the localhost on which means the local server MariaDB is running. The next specifies a single host. The third uses % to specify any subdomain of the example.net domain.

```
CREATE USER 'tom'@ 'localhost' IDENTIFIED BY 'retail';
CREATE USER 'richard'@ 'powr.example.net' IDENTIFIED BY 'nuclear';
CREATE USER 'robert'@ '%.example.net' IDENTIFIED BY 'pilot';
```

Instead of hostnames we can also use IP addresses as shown in the following three examples. The first has an exact IP address identifying a single computer. The second uses a % sign in the last quad of the IP address so any computer where the first three sets of numbers in the IP address match will be able to connect. The third uses a subnet mask, but the end result (in this example at least) is the same as the second.

```
CREATE USER 'dallin'@ '192.168.1.1' IDENTIFIED BY 'judge';
CREATE USER 'russell'@ '192.168.1.%' IDENTIFIED BY 'surgeon';
CREATE USER 'russell'@ '192.168.1.0/255.255.255.0' IDENTIFIED BY
'business';
```

One benefit to using IP addresses instead of domain names is that no name resolution or domain validation needs to be made. Such system calls to lookup and check the validity of domains can be costly and might take time and resources better spent on other things. To enforce a no domain names policy, add `skip-name-resolve=1` to the `[mysqld]` section of the `my.cnf` or `my.ini` file.

Complete documentation of the `CREATE USER` statement is available at <https://mariadb.com/kb/en/create-user/>.

Granting permissions

By default, new users do not have permission to do anything except logging in, which is not very useful. So the next thing we need to do is give them the permissions they need. This is done using the `GRANT` statement. Using this statement, we will be able to `GRANT` users the appropriate permissions. The `GRANT` statements have the following basic pattern:

```
GRANT <privileges> ON <database> TO <user>;
```


We customize the <privileges>, <database>, and <user> parts as needed. The <user> section should match the 'username'@'host' part of the CREATE statement, otherwise, we'll be creating what is essentially a new user. We can also add an IDENTIFIED BY 'password' section to the end of the GRANT statement if we want to change the password (or add a password to an account that doesn't have one).

Here are some examples. This first one grants all privileges including the grant option on all databases and the user can log in from anywhere. We should not often set up users with such broad authority, and when we do we need to make sure, we use an appropriate CREATE USER statement first and assign the user a password (or assign the password here). If the user doesn't exist, the GRANT statement will create one, but if the user doesn't exist and our GRANT statement doesn't include an IDENTIFIED BY 'password' section then the user will be created without a password, so it's a good habit to first create the user with a password, and then grant the user the rights they need.

```
GRANT ALL ON *.* TO 'robert'@'%' WITH GRANT OPTION;
```

The following example is a standard set of permissions for a regular user who needs read and write access to a database called `serv`. If a user just needs read access, we can just assign the user the SELECT privilege. By specifying `serv.*` as the database, the user only has these rights on tables in the `serv` database. Multiple privileges are separated by commas.

```
GRANT SELECT,INSERT,UPDATE,DELETE ON serv.* TO 'jeffrey'@'localhost';
```

This following user has read access (SELECT) to just the `staff` table in the `edu` database, and the user has the GRANT OPTION privilege so they can grant that same right to other users.

```
GRANT SELECT ON edu.staff TO 'david'@'localhost' WITH GRANT OPTION;
```

The following example gives a user all rights on the `logan` database. We'll also limit this user to 100 queries per hour, just because we can.

```
GRANT ALL ON logan.* TO 'quentin'@'localhost' WITH MAX_QUERIES_PER_HOUR  
100;
```

Complete documentation of the GRANT statement is available at
<https://mariadb.com/kb/en/grant/>.

Adding and removing privileges

Sometimes it becomes necessary to remove a privilege or two from a user, or to give them more privileges. Giving additional privileges is easy, just run an additional `GRANT` statement with the new rights and they will be added. To remove privileges, we use the `REVOKE` statement. It has the following pattern:

```
REVOKE <privileges> ON <database> FROM <user>;
```

To remove a `GRANT OPTION` privilege, specify it in the privileges section. The following example removes the `DELETE` and `GRANT OPTION` permissions from the `todd` user:

```
REVOKE DELETE,GRANT OPTION ON cust.* FROM 'todd'@'%';
```

To remove all privileges from a user ('`neil'@'%.example.com'` in this example), we use the following special command:

```
REVOKE ALL,GRANT OPTION FROM 'neil'@'%.example.com';
```

We, of course, need to customize the user part to match the user for whom we are removing privileges. The previous statement is special in that it must be used as written even if the user doesn't have the `GRANT OPTION` privilege. If we remove the `GRANT OPTION` privilege from it the statement won't run.

Complete documentation of the `REVOKE` statement is available at: <https://mariadb.com/kb/en/revoke/>.

Showing grants

To show the grants available for a user, we use the `SHOW GRANTS` command. It has the following pattern:

```
SHOW GRANTS FOR <user>;
```

All we have to do is customize the `<user>` part with the information of the user we want to look at. Here is an example:

```
SHOW GRANTS FOR 'dieter'@'10.2.200.4';
```

The output of the `SHOW GRANTS` command is a `GRANT` statement that encapsulates all of the user's privileges. This is useful if you want to give another user the exact same privileges. For example, the output of the preceding `SHOW GRANTS` command might be as follows:

```
+-----+
| Grants for dieter@10.2.200.4 |
+-----+
| GRANT ALL PRIVILEGES ON *.* TO 'dieter'@'10.2.200.4' |
+-----+
```

Changing passwords

To change the password of a user, we use the `SET PASSWORD` statement. It has the following pattern:

```
SET PASSWORD FOR <user> = PASSWORD('<password>');
```

Here is an example:

```
SET PASSWORD FOR 'henry'@'%' = PASSWORD('niftypassword');
```

Complete documentation of the `SET PASSWORD` statement is available at <https://mariadb.com/kb/en/set-password/>.

Removing users

To remove a user completely, we use the `DROP USER` statement. It has the following pattern:

```
DROP USER <user>;
```

Here is an example:

```
DROP USER 'tom'@'%';
```

When a user is dropped, all grants are automatically removed.

Complete documentation of the `DROP USER` statement is available at <https://mariadb.com/kb/en/drop-user/>.

Summary

In this chapter, we learned about adding and removing users and how to give those users the permissions they need or take them away as needed. Up till now, we've only talked about things related to our databases (securing them, managing users, and so on). We haven't actually done anything with the actual data – you know the stuff that databases are good at storing and retrieving. Well, the time to talk about it is now. In the next chapter, *Using MariaDB*, we'll learn some basic SQL commands that we can use with the `mysql` command-line client program to create databases, insert data, read data, and so on.

5

Using MariaDB

This chapter will focus on using the command-line `mysql` client to perform common tasks. We'll learn about creating and dropping databases and tables, making modifications to our tables, inserting and updating data, viewing the results, and more. When finished we will have a good grasp of the basics. We'll know our way around, a skill which is always useful, even if we end up spending most of our time using graphical point-and-click clients, or applications that interact with the database for us.

Running the `mysql` command-line client

A big part of becoming a MariaDB expert is learning how to effectively and efficiently use the command-line `mysql` client program. Many MariaDB users interact with the server using custom programs that have been developed specifically for or by them. At a lower level though, every interaction these applications have with MariaDB can be done with the command-line client.

MariaDB has a client-server architecture, which means there are two parts to it: the server, which is the part that does the heavy behind-the-scenes stuff, and the client, which is the part we use to access and interact with the server. We hardly ever interact directly with the server part. There are many different clients for MariaDB, but only one is maintained by the MariaDB developers and included with every copy of MariaDB—the `mysql` command-line client.

To start the client, we open up a command-line or terminal window and type `mysql` with some options and press *Enter*. The basic syntax is as follows:

```
mysql [-u <user>] [-p] [-h <host>] [<database>]
```

All of the options in the previous example are in brackets ([]) to show that they are all optional. The parts in angle brackets (< >) are bits that we must supply if we choose to use that option. For example, if we use the -u option, we must supply a username.

Most of the time, we will use the user (-u) and password (-p) options. We will also often specify the database that we want to connect to when the client launches. When we connect remotely to a MariaDB server on another computer, we will use the host (-h) option.

A successful connection will look similar to the following:

```
daniel@gandalf ~ $ mysql -u root -p
Enter password:
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 31
Server version: 5.5.30-MariaDB-mariadb1-quantal-log mariadb.org binary
distribution
Copyright (c) 2000, 2013, Oracle, Monty Program Ab and others.
Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.
MariaDB [(none)]>
```

The last line of the output, MariaDB [(none)]>, is the MariaDB prompt. It appears whenever MariaDB is waiting for us to give it a command. Apart from its primary purpose, the prompt gives us two pieces of very useful information. First, the prompt says MariaDB which tells us that we are connecting to an actual MariaDB database server (as opposed to a compatible database server that isn't actually MariaDB). Second, the part in brackets tells us which database on the server we are currently using; in this case, we aren't using any database, so it says (none).

Using a database

We generally want to be connected to a specific database when we use the command-line client. To use a database, we either specify it on the command-line when launching the client as shown in the previous section, or we use the USE command, to tell the client which database we want to talk to. The following example illustrates connecting to a database named test. Notice that the prompt changes to let us know the name of the database it is currently connected to.

```
MariaDB [(none)]> USE test;
Database changed
MariaDB [test]>
```

If the database does not exist when we try to USE it, we will get the following error.

```
MariaDB [(none)]> use test1;
ERROR 1049 (42000): Unknown database 'test1'
```

Listing all databases on a server

To show a list of all of the databases on a server, use the `SHOW DATABASES` command as in the following example:

```
MariaDB [(none)]> SHOW DATABASES;
+-----+
| Database          |
+-----+
| information_schema |
| dbt3_s001         |
| flightstats       |
| mysql             |
| performance_schema |
| test              |
+-----+
6 rows in set (0.00 sec)
MariaDB [(none)]>
```



You may have noticed in the previous examples that all the commands ended with a semi-colon (;). This is called the **delimiter** and it is a characteristic feature of **Structured Query Language (SQL)**. We interact with the command-line client using this language. In basic terms, SQL is a computer language optimized for interacting with a database. MariaDB uses its own variant of SQL which is similar to but not exactly the same as the SQL variants used by other databases. When we learn how to write SQL statements for MariaDB, we also learn a good deal about writing SQL for other databases, but there are some differences. For instance, `USE` and `SHOW` are commands, which exist in MariaDB but not in some other databases that use their own variety of SQL.

Creating and dropping databases

When we install MariaDB, we're installing a database server, not a specific database, and a single MariaDB database server can have several databases inside it. Here's an analogy that can help us understand this arrangement: a database can be thought of as a large filing cabinet. The filing cabinet contains a number of drawers and inside each drawer are files with information. In this analogy, the filing cabinet is a database, the drawers are tables within the database, and the files are rows of data within the tables. So what is MariaDB? It's the room the filing cabinet is located in, and it's a large room so we can put many filing cabinets inside it. When MariaDB is installed, the installer creates a system database that MariaDB uses to keep track of users and databases and other housekeeping information. The installer also creates a test database for experimentation and learning and a read-only database where MariaDB stores performance statistics. We don't want to use the system database as we could mess up the entire server if we made a mistake. We can't put data into the statistics database, called `performance_schema`, because it is read only. We can use the test database, but we probably don't want to use it for anything permanent. So one of our first tasks when we start using MariaDB is to create at least one database for us to use.

Generally, databases are created for specific things or specific applications. For example, we could have an accounting database for the finance department, a human resources database for the HR department, and a parts database for the warehouse.

Creating and dropping (deleting) databases are two things that we will do less often than just about anything else when working with MariaDB. There just isn't much call for it in day-to-day work. We generally create a database and then use it as long as it is needed (which could be for years or decades) and then we delete (drop) it. Thankfully, the commands for creating and dropping a database are very simple, so they're easy to remember.

Creating a database

As mentioned previously, creating a database is not something we do often. To create a database in MariaDB, we use the `CREATE DATABASE` command. The basic syntax is as follows:

```
CREATE DATABASE <dbname>;
```

If the database already exists when we try to create it, we will receive an error. We can suppress the error with `IF NOT EXISTS`.

Here are some examples:

```
CREATE DATABASE my_database;  
CREATE DATABASE IF NOT EXISTS my_database;
```

The previous two commands are equivalent if the database does not exist. If the database does exist, the first command will exit with an error and the second command will do nothing.

Full documentation of the `CREATE DATABASE` command is available at <https://mariadb.com/kb/en/create-database/>.

Dropping a database

As mentioned before, it isn't often that we need to remove or delete a database, but when we do, we use the `DROP` command. The basic syntax is as follows:

```
DROP DATABASE <databasename>;
```

If the named database doesn't exist when we try to drop it, we will receive an error. We can suppress the error with `IF EXISTS`.

Here are a couple of examples that drop the database we just created:

```
DROP DATABASE my_database;  
DROP DATABASE IF EXISTS my_database;
```

The preceding two commands are equivalent if the database `my_database` exists. If the database does not exist, the first command will exit with an error and the second command will do nothing.



When dropping a database, user privileges for the database are not removed. We need to revoke them manually, or drop the user entirely; otherwise, if or when the database is recreated the user will still have the privileges. See *Chapter 4, User Account Management*, for information on managing users and their privileges.

Complete documentation of the `DROP DATABASE` command is available at <https://mariadb.com/kb/en/drop-database/>.

Creating, altering, and dropping tables

Databases contain tables. Tables are two-dimensional data structures containing rows and columns. A row corresponds to a single record in a database and records are divided into columns. Think of database tables like a specialized spreadsheet.

The many multiple tables in a MariaDB database have columns, which can relate in one way or another. For example, the `id` column in an employee table may relate to the `employee_id` column in an address table. These relationships (also called **foreign keys**) are why we call MariaDB a **relational** database.

A database without tables of data is nothing more than a name. Until we create some tables and start adding data to those tables, our database is empty and useless.

Creating a table

There are few things in MariaDB we will spend more time on, at least in the beginning than when we do creating or defining the tables for our database.

We use the `CREATE TABLE` command to create tables. Using the command, we define the structure of the table. The structure includes such things as the number of columns and the type of data that we want to store in each column. Datatypes include things such as numbers, text, and dates. For example, if we are creating an employee table, we might decide to store an employee ID number (number), last name (text), given names (text), preferred name (text), birthdate (date), and so on.

We might also want to store the e-mail addresses, phone numbers, and home addresses of the employees, but we don't want to store duplicate data, or define extra columns that are hardly ever used, so we should put those in separate tables and then link the records back to the appropriate employee. We do this because people often have multiple phone numbers and e-mail addresses, and sometimes even extra home addresses and we might want to store all of them.

The process by which we refine our table definitions is called **normalization**. There isn't space here for a complete discussion of this process, but the MariaDB Knowledge Base has a page which discusses it in depth available at <https://mariadb.com/kb/en/recap-the-relational-model>.

For a basic database for an online store, we might have tables for customers, products, orders, product reviews, customer addresses, and more. We can create as many tables as we need, but as mentioned previously, we should give the design some thought so that we don't store duplicate or unused data. That said, don't worry too much, we can always make changes after the fact with the `ALTER TABLE` command (see the *Altering a table* section in this chapter).

The basic syntax of the `CREATE TABLE` command is as follows:

```
CREATE TABLE table_name (<column_definitions>;
```

As with creating a database, we can add an `IF NOT EXISTS` command before the table name to suppress the error that would appear if the table exists when we try to create it.

The `<column_definitions>` part has the following basic pattern:

```
<column_name> <data_type>
[NOT NULL | NULL]
[DEFAULT <default_value>]
[AUTO_INCREMENT]
[UNIQUE [KEY] | [PRIMARY] KEY]
[COMMENT '<string>']
```

The parts in angle brackets (`<>`) are the bits that we fill in. The parts in square brackets (`[]`) are optional and the pipe character (`|`) means *or*. For example, we can (but do not have to) specify `NULL` or `NOT NULL` in a single column definition but we cannot specify both. Columns are allowed to be `NULL`, or have no value, by default. Marking a column as `NOT NULL` means it can never be empty; some value has to be assigned to it. Multiple column definitions are separated by commas.

There are many different datatypes (`<data_type>`) to choose from. A datatype (`<data_type>`) is a type of data being stored. They exist because each is efficiently stored somewhat differently. Plain numbers can be treated differently than dates and vice versa. Common ones include numeric, string, and date datatypes. Numeric datatypes include `INTEGER` (commonly written as `INT`), and `FLOAT` (for floating point numbers). String (or text-based) datatypes include `CHAR`, `TEXT`, and `VARCHAR`. Lastly, date and time datatypes include `DATE`, `TIME`, and `DATETIME`.

See a complete list of supported datatypes at <https://mariadb.com/kb/en/data-types/>.



Don't worry about trying to memorize all of the different datatypes now. They'll become second nature as we gain experience using MariaDB.

After specifying the type, length, and precision (for some datatypes), we specify other options. We can specify whether or not the column is allowed to be empty (or `NULL`), what the default value (`<default_value>`) is, if anything, whether the column auto-increments (only for numeric datatypes), whether the value in the column should be `UNIQUE`, whether the column is a primary key, and a comment, if desired.

A primary key is a column or group of columns, which uniquely identifies a specific row in the table. No other row in a given table is allowed to have the same primary key. If we try to input a row with a primary key that matches another primary key in the table we will get an error.

For our preceding `employees` example, we might use the following `CREATE` statement to create the table (use the `test` database or `CREATE` a new database and then `USE` it if you want to follow along):

```
CREATE TABLE employees (  
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,  
    surname VARCHAR(100),  
    givenname VARCHAR(100),  
    pref_name VARCHAR(50),  
    birthday DATE  
);
```

When we run the preceding code, the output looks like this:

```
Query OK, 0 rows affected (0.00 sec)
```

A result of `Query OK` means that the table was created successfully. Zero rows were affected because this brand new table has no data in it yet.

Full documentation of the `CREATE TABLE` command can be found at <https://mariadb.com/kb/en/create-table/>.

Showing the command used to create a table

At any time, for example, if we want to create a similar table in a different database, we can use the `SHOW CREATE TABLE` command to show a command that will recreate the table exactly. Here is an example:

```
MariaDB [test]> SHOW CREATE TABLE employees\G  
***** 1. row *****  
Table: employees  
Create Table: CREATE TABLE `employees` (  
  `id` int(11) NOT NULL AUTO_INCREMENT,  
  `surname` varchar(100) DEFAULT NULL,  
  `givenname` varchar(100) DEFAULT NULL,  
  `pref_name` varchar(50) DEFAULT NULL,  
  `birthday` date DEFAULT NULL,
```

```
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```



The \G at the end of the command in this example is an alternative to using a semicolon (;) and when used it presents the output in a slightly different way which works well for this example.

The actual `CREATE TABLE` command that is displayed is slightly different from the `CREATE TABLE` command that we used to create it, but the table created is exactly the same. The differences exist because MariaDB is giving us enough information to recreate the table exactly, even if we're creating it on a different server with different defaults.

For example, the `ENGINE` and `DEFAULT CHARSET` parts after the column definitions are default table options on this MariaDB Server. Here they are specified because on a different MariaDB server the defaults may be different.

Full documentation of the `SHOW CREATE TABLE` command can be found at <https://mariadb.com/kb/en/show-create-table/>.

Exploring the structure of a table

If we don't necessarily want to look at the commands used to create a table but we want to know the structure of a table, we can use the `DESCRIBE` command as follows:

```
MariaDB [test]> DESCRIBE employees;
```

```
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| id         | int(11)       | NO   | PRI | NULL    | auto_increment |
| surname    | varchar(100)  | YES  |     | NULL    |                |
| givenname  | varchar(100)  | YES  |     | NULL    |                |
| pref_name  | varchar(50)   | YES  |     | NULL    |                |
| birthday   | date          | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.03 sec)
```

Full documentation of the `DESCRIBE` command can be found at <https://mariadb.com/kb/en/describe/>.

Altering a table

We can spend hours, days, and even weeks getting our tables defined just the way we want them, but chances are at some point we'll need to make some changes. This is where the `ALTER TABLE` command comes into play.

The basic syntax for the `ALTER TABLE` command is as follows:

```
ALTER TABLE table_name <alter_definition>[, alter_definition] ...;
```

An `<alter_definition>` command can `ADD`, `MODIFY`, and `DROP` (delete) columns from tables, among other things. Multiple alter definitions in a single `ALTER TABLE` command are separated by commas. Because we can have multiple alter definitions in one `ALTER TABLE` command, the syntax examples in the next four sections will not contain the beginning `ALTER TABLE table_name` part that must begin an `ALTER TABLE` command. The examples that demonstrate actual usage will.

Adding a column

An `<alter_definition>` attribute of an `ALTER TABLE` command is used to add a column has the following pattern:

```
ADD <column_name> <column_definition> [FIRST | AFTER <column_name>]
```

The `FIRST` and `AFTER` parts are optional. We can use one, but not both. The `FIRST` option puts the new column as the first column of the row. The `AFTER` option lets us specify, which column the new column appears after.

For example, the following will create a new `username` column and place it after the `pref_name` column:

```
ALTER TABLE employees ADD username varchar(20) AFTER pref_name;
```

Modifying a column

An alter definition of an `ALTER TABLE` command to modify a column has the following pattern:

```
MODIFY <column_name> <column_definition>
```

For example, the following `ALTER TABLE` command will change the `pref_name` column to `varchar(25)`:

```
ALTER TABLE employees MODIFY pref_name varchar(25);
```

Dropping a column

An alter definition of an `ALTER TABLE` command to delete a column has the following pattern:

```
DROP <column_name>
```

For example, the following `ALTER TABLE` command will delete the `username` column we created earlier:

```
ALTER TABLE employees DROP username;
```

Full documentation of the `ALTER TABLE` command is found at <https://mariadb.com/kb/en/alter-table>.

Dropping a table

When we no longer need a table, just as when we no longer need a database, we drop it. Out of the table commands, this one is by far the easiest. The basic syntax of the command is:

```
DROP TABLE <table_name>
```

If we try to drop a table that doesn't exist we will receive an error. We can suppress the error with `IF EXISTS`.

Here are a couple of examples:

```
DROP TABLE mytable;
```

```
DROP TABLE IF EXISTS mytable;
```

If the table exists, the preceding two commands are equivalent. If the table doesn't exist the first command will exit with an error and the second command will do nothing.

Full documentation of the `DROP TABLE` command can be found at <https://mariadb.com/kb/en/drop-table>.

Selecting, inserting, updating, and deleting data

Getting data into and out of our database tables and updating, and deleting data when necessary is where we will spend most of our time when working with MariaDB.

Inserting data

To put data into our database, we use the `INSERT` command. The basic syntax is as follows:

```
INSERT [INTO] <table_name> [( <column_name>[, column_name,...])]
{VALUES | VALUE}
({expression|DEFAULT},...) [, (...),...];
```

As with the `CREATE TABLE` command, the parts in angle brackets (`<>`) are what we'll replace with our own values. The square brackets (`[]`) are optional and the pipe character (`|`) means *or*. The curly brackets (`{ }`) specify a section that is mandatory but for this there is a choice of the key word you can use. For example, the `INTO` keyword is optional but makes the `INSERT` line more readable and we can use the keyword `VALUES` or `VALUE` depending on whether or not we are inserting a column of information or multiple items but we must use one of them. Anywhere when we see three dots (`. . .`) it represents a part where the previous part can be repeated.

Expression just means the value we want to put in the column. It could be a calculated value (such as today's date + four days), a static value (such as John) or it could be the default value assigned to the column (if it has one). Default values are assigned using the keyword `DEFAULT` without any quotes.

If we do not specify the columns we want to insert into, then we must specify a value for each column. If we don't want to assign a value (and if the column definition allows it) we can specify `NULL`, which means no value.

We can also insert multiple rows at once (remember to use the keyword `VALUES` when doing so!). Each row is wrapped in parentheses and multiple rows are separated by commas.

Following are some basic `INSERT` examples. They are split onto multiple lines to aid reading but can be entered with a single long line if desired. Remember that a single command ends with either the semicolon (`;`) or `\G` delimiters.

```
INSERT INTO employees (surname,givenname) VALUES
    ("Taylor","John"),
    ("Woodruff","Wilford"),
    ("Snow","Lorenzo");
```

```
INSERT INTO employees (pref_name,givenname,surname,birthday)
    VALUES ("George","George Albert","Smith","1970-04-04");
```

```
INSERT employees (surname) VALUE ("McKay");
```

```
INSERT INTO employees VALUES
    (NULL, "Kimball", "Spencer", NULL, NULL);
```

Full documentation of the INSERT command is found at <https://mariadb.com/kb/en/insert/>.

Updating data

When data in a table needs to be updated, we use the UPDATE command. The basic syntax is as follows:

```
UPDATE <table_name>
    SET column_name1={expression|DEFAULT}
    [, column_name2={expression|DEFAULT}] ...
    [WHERE <where_conditions>];
```

Unlike the INSERT command, when we are updating data we specify the data we want to insert right after each column name. Another difference is inclusion of a WHERE section. The WHERE section is very important because we use it to specify the exact column or columns of data in the table we want to change. If we omit the WHERE section the UPDATE statements will update every instance of that column. For example, we could accidentally change every employee's phone number to the same number when all we wanted was to update Gordon's.

One thing we should do in our example employees table is add birthdays and preferred names for some of our employees:

```
UPDATE employees SET
    pref_name = "John", birthday = "1958-11-01"
    WHERE surname = "Taylor" AND givenname = "John";
UPDATE employees SET
    pref_name = "Will", birthday = "1957-03-01"
    WHERE surname="Woodruff";
UPDATE employees SET
    birthday = "1964-04-03"
    WHERE surname = "Snow";
```

For each of the preceding commands, MariaDB should output the following two lines (the amount of time, 0.03 seconds in the example, may be different):

```
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

What these two lines are telling us is that one row matched our `WHERE` section and that one row was changed.

Full documentation of the `UPDATE` command is found at <https://mariadb.com/kb/en/update/>.

Deleting data

When data needs to be removed from a database table, we use the `DELETE` command. The basic syntax is as follows:

```
DELETE FROM <table_name> [WHERE <where_conditions>];
```

As with `UPDATE` statements, the `WHERE` part of a `DELETE` statement is optional, but if we leave it off, the command will delete every row in the table, which is even more catastrophic than leaving off the `WHERE` part in an `UPDATE` statement, if such a thing is possible, so make it a habit to always include it.

As an example, let's delete the Spencer Kimball employee:

```
DELETE FROM employees
WHERE givenname="Spencer" AND surname="Kimball";
```

Full documentation of the `DELETE` command is found at <https://mariadb.com/kb/en/delete/>.

Reading data

The command for reading data from our database is called `SELECT`. Of all SQL commands, this is the one we will probably use most often. As such the syntax is rather complex, or can be, if we choose to use all the various options. The basic syntax is as follows:

```
SELECT <what> FROM <table_name>
    [WHERE <where-conditions>]
    [ORDER BY <column_name>];
```

A common <what> condition is to grab everything. This is done using an asterisk (*) as follows:

```
MariaDB [test]> SELECT * FROM employees;
```

id	surname	givenname	pref_name	birthday
1	Taylor	John	John	1958-11-01
2	Woodruff	Wilford	Will	1957-03-01
3	Snow	Lorenzo	NULL	1964-04-03
4	Smith	George Albert	George	1970-04-04
5	McKay	NULL	NULL	NULL

```
5 rows in set (0.00 sec)
```

However, when our table contains lots of columns, we probably will only want to select certain of them. To do so we list the columns, separated by commas, in the <what> section. For example, we could select just the preferred names and surnames as follows:

```
MariaDB [test]> SELECT pref_name,surname FROM employees;
```

pref_name	surname
John	Taylor
Will	Woodruff
NULL	Snow
George	Smith
NULL	McKay

```
5 rows in set (0.00 sec)
```

Another thing we may want to do is select only certain rows, for example, everyone born after 1 January, 1960:

```
MariaDB [test]> SELECT * FROM employees
-> WHERE birthday > '1960-01-01';

+----+-----+-----+-----+-----+
| id | surname | givenname | pref_name | birthday |
+----+-----+-----+-----+-----+
| 3 | Snow | Lorenzo | NULL | 1964-04-03 |
| 4 | Smith | George Albert | George | 1970-04-04 |
+----+-----+-----+-----+-----+

2 rows in set (0.00 sec)
```

The > sign is a comparison operator. Just like in math, it means greater than. There are many other comparison operators. See <https://mariadb.com/kb/en/comparison-operators/> for a complete list.



The arrow (->) in the previous output example is not something we typed. The mysql command-line client program inserted it to show that we pressed the *Enter* key before ending our command with a semicolon (;), and so the command we are entering is continuing on a second line. If we pressed *Enter* and just forgot to end our command, we can just type a semicolon and press *Enter* again. In the previous example, I did it on purpose to split the command onto two lines to make it easier to read.

Our output, by default, is sorted based on the order in which it was inserted into the database. For a list of employees we really should sort on the surname column. To do this we use an ORDER BY clause as follows:

```
MariaDB [test]> SELECT * FROM employees ORDER BY surname;

+----+-----+-----+-----+-----+
| id | surname | givenname | pref_name | birthday |
+----+-----+-----+-----+-----+
| 5 | McKay | NULL | NULL | NULL |
| 4 | Smith | George Albert | George | 1970-04-04 |
| 3 | Snow | Lorenzo | NULL | 1964-04-03 |
| 1 | Taylor | John | John | 1958-11-01 |
| 2 | Woodruff | Wilford | Will | 1957-03-01 |
+----+-----+-----+-----+-----+

5 rows in set (0.00 sec)
```

We can set the order of any column the way we wish using `ORDER BY`, and even specify multiple columns, separating each column with a comma just as we do in the `<what>` section.

We can also use pattern matching to select rows when we only know some of the information. For example, suppose we want to look up an employee whose name begins with the letters `McK` but we don't remember the rest. We can look up this employee using a `WHERE` clause as follows:

```
MariaDB [test]> SELECT * FROM employees
-> WHERE surname LIKE "McK%";
```

id	surname	givenname	pref_name	birthday
5	McKay	NULL	NULL	NULL

```
1 row in set (0.00 sec)
```

The percent sign (%) is what is known as a **wildcard character**. It matches zero or more instances of any character or group of characters. So by specifying the surname pattern `McK%`, we are saying that we want any surnames that begin with those letters and are followed by zero or more other characters. This pattern would match items such as `McKay`, `McKinsey`, `McKool`, and even `McK`.

Full documentation of the `SELECT` command is found at <https://mariadb.com/kb/en/select/>.

The `SELECT` commands are even more powerful when we start using `JOINS` to gather data from multiple tables. For example, if we had an address table along with our employee table, we could `SELECT` the names and addresses of all employees who live in a certain city and whose birthdays are less than a month away so that we can send them a gift certificate to a restaurant in that city. Full documentation of `JOIN` syntax is found at <https://mariadb.com/kb/en/join/>

Summary

In this chapter, we learned some basic commands which are useful when using MariaDB and connecting with the `mysql` command-line client program. We've also become familiar with the basic SQL database **Create, Read, Update, and Delete (CRUD)** operations that will be part of nearly all of our interactions in MariaDB.

In the next chapter, *MariaDB Maintenance*, we will learn how to keep our database running smoothly.

6

MariaDB Maintenance

Just like houses and cars, databases need to be maintained if they are to run smoothly. In this chapter we'll cover a variety of maintenance-related topics. We'll start with an exploration of the various MariaDB log files, which tell us what MariaDB is doing. Then we'll move on to a discussion about optimizing MariaDB. We'll then discuss what is probably the most important part of maintaining our database: backups. Lastly, we'll cover the utilities MariaDB includes to repair tables.

MariaDB log files

Depending on how we configure it, MariaDB will keep very detailed or very sparse logs. The location of these logs is configured in our `my.cnf` (`my.ini` on Windows) MariaDB configuration file. On Linux, the default location is `/var/log/mysql/` and on Windows the default location is in the MariaDB data directory.

There are several different kinds of logs, each kind serving a different purpose.

The all-important binary log

The MariaDB binary log is a series of files that contain events. An **event** is a description of a modification to the contents of our database. As indicated by the name, and unlike most other kinds of log files, MariaDB binary log files are in a binary format. They are not readable by us unless we use a helper program such as `mysqlbinlog`.

The binary log is controlled by the `log_bin` variable. The main purpose of the variable is to turn binary logging on and off. Basically, if the variable is present in the `[mysqld]` section of our config file, binary logging will be turned on, and if it isn't, binary logging will be turned off. An optional function of this variable is to set the name and location of the binary log. Here is an example:

```
log_bin = /var/log/mysql/mariadb-bin
```


MariaDB will take the name and add numbers to the end of the actual files it writes to. Here is an example of using the `mysqlbinlog` program to display the contents of a binary log in human-readable text:

```
mysqlbinlog /var/log/mysql/mariadb-bin.000269
```

Each event in a binary log file is preceded with some comment lines that give the date and time of the event and its position in the log.

Apart from its informational value, the MariaDB binary log has some other uses. First, it can be used to recover after a server crash. It is also used when replicating from one server to another. When used for replication, they are transferred to the slave servers as relay logs, but they are basically the same as regular binary logs and can be read with the `mysqlbinlog` program.

More information about the binary log can be found at <https://mariadb.com/kb/en/binary-log/>.

The error log

The error log is where MariaDB logs information about critical errors. This is also where MariaDB records startup and shutdown information. If MariaDB crashes or fails to start, this is the log where we should look first.

We can control the location using the `log_error` variable, which, like other logging variables, is placed in the `[mysqld]` section of our configuration files. Here is an example:

```
log_error = /var/log/mysql/error.log
```

The default configuration files shipped with MariaDB on Linux configure this file to be at this location, but it can be placed elsewhere.

Unlike the binary log, the absence of this variable does not turn error logging off. If the location is not configured in our `my.cnf` or `my.ini` file, error logging is still enabled and the default location is the data directory. Also, the default name will be `hostname.err` where `hostname` is the name of the computer MariaDB is running on.

More information about the error log can be found at <https://mariadb.com/kb/en/error-log/>.

The general query log

The general query log is disabled by default. This is because the general query log, when enabled, stores a record of every query MariaDB receives, including queries that don't change any data. This is in contrast to the binary log, which only stores queries that change data. On a busy server with lots of users, storing all queries can lead to lots of huge log files very quickly. It is not usually necessary or recommended to enable this log.

However, if we are trying to discover a hidden performance bottleneck or absolutely need a record of everything the server is doing, this log can be enabled with the `general_log` and `general_log_file` variables. The first is used to explicitly turn the general log on (`=1`) or off (`=0`). The second configures where we want the log to be and what we want it to be called. Here is an example:

```
general_log = 1
general_log_file = /var/log/mysql/mysql.log
```

If we are only interested in queries which change data, the only log we need to look at is the binary log and we do not need to enable the general query log. As it says in a comment in the default `my.cnf` file shipped with MariaDB, the general log is a *performance killer*, so we should only enable it if we need to, and only for short periods of time.

More information about the general query log can be found at <https://mariadb.com/kb/en/general-query-log/>.

The slow query log

The MariaDB slow query log, when enabled, contains a record of queries that take longer than a configured amount of time to run. This log is very useful when tuning and optimizing MariaDB as it stores lots of useful information such as the time it took to execute a long query, the user who executed the query, the hostname the user came from, and other details.

This log is disabled by default. To enable it we add the following to the `[mysqld]` section of our `my.cnf` or `my.ini` config file:

```
slow_query_log = 1
```

There are four other variables that we use to control the behavior of the slow query log. We use the `slow_query_log_file` variable to set the location of the log file. We use the `long_query_time` variable to set the amount of time the query has to run before MariaDB considers it to be slow. Times can be expressed in whole seconds down to microsecond precision (`0.000001`).

The `log_slow_rate_limit` variable is used to control how often long queries are actually logged. For example, a setting of 20 would log every twentieth query, or 5 percent of the slow queries. This is useful if our slow query log is growing too fast. If this variable is not present, the default is for the slow query log to log every slow query.

Lastly, the `log_slow_verbosity` variable controls what information is logged. Possible values include:

- `microtime`: Makes the log to log the queries in microseconds
- `query_plan`: Logs query execution plan information
- `innodb`: Adds additional statistical information about queries that touch XtraDB and InnoDB tables
- `standard`: Turns on both the `microtime` and `innodb` variables
- `full`: Turns on all options
- `profiling`: Allows the logged queries to be profiled

We can specify multiple options if we separate them with commas. Here is an example which turns on the slow query log and sets some common options:

```
slow_query_log = 1
slow_query_log_file = /var/log/mysql/mariadb-slow.log
long_query_time = 0.05
log_slow_rate_limit = 30
log_slow_verbosity = query_plan,standard
```

More information about the slow query log can be found at <https://mariadb.com/kb/en/slow-query-log/>.

Optimizing and tuning MariaDB

The subject of tuning and optimizing MariaDB, and the applications that connect to it, for maximum performance is worthy of a book in itself. We won't go into the details of specific strategies here because it's generally not necessary when we're just getting started with MariaDB. It is useful to know a little about the subject though, which is explained in the following section.

The basic process of tuning and optimizing MariaDB starts with identifying the choke points, that is, the places that are causing unnecessary slowdowns. Using the slow query log to identify these choke points is a good place to start.

Once a problem query (or set of queries) has been identified, the next step is to implement a fix of some sort. This could be as simple as rewriting the query to be more efficient. Or it could be sped up by adding an index to the table.

We can also gather and examine user and table statistics to identify patterns of usage that we can potentially optimize. Or we can examine our table definitions to see if there are any tweaks that can be made there to make things faster or more efficient. The list of ways to squeeze more performance out of MariaDB goes on and on.

If the query and our database are as optimized as we can make them, there are still things that we can do. Hardware, for example, can be a limitation. A busy database that needs to respond quickly needs to be on fast hardware. Fast disks, lots of memory, and a fast processor are all important.

More information on optimizing and tuning MariaDB can be found at <https://mariadb.com/kb/en/optimization-and-tuning/>.

Backing up MariaDB

MariaDB ships with a couple of utilities that can be used to back up our databases. Data in MariaDB is written to special files on disk, so it may be tempting to think that we can just backup the MariaDB data directory and be done. The problem with this is that the data files are always open and in use while MariaDB is running and problems can arise if we try to back up the files directly.

Basic backups with mysqldump

By default, the `mysqldump` client backup utility generates SQL backups. These backups contain all the necessary SQL commands to recreate tables and restore the data in those tables.

There are many options, but the basic syntax is as follows:

```
mysqldump [-u username] [-p] database_name [table_name]
```

If `table_name` is not given, `mysqldump` will backup all the tables in the named database. For example, the following command will backup the entire `test` database:

```
mysqldump -u root -p test > test.sql
```

The output of `mysqldump` goes to standard out. When running the command from a terminal, it will be echoed directly to the screen. So in the preceding example command, we use the `>` redirect character to direct the output into a file named `test.sql`.

To restore the preceding backup, we can use the `mysql` command-line client as follows:

```
mysql -u root -p test < test.sql
```

As with the `mysqldump` example, we use a redirect character, but this time it is redirecting in the opposite direction, from the `test.sql` file to the `mysql` client. The `mysql` client reads the file and executes all of the SQL commands, restoring the backed up tables and their data.

We can also use `mysqldump` to create tab-delimited files. This is done using the `--tab` option. When using this option, `mysqldump` will create two files. A `tablename.sql` file with the SQL commands to recreate the table, and a `tablename.txt` file with the actual data in tab-delimited format. Here is an example using `mysqldump` and `--tab` to backup up the `employees` table in our test database:

```
mysqldump --tab /tmp/ -u root -p test employees
```

The `--tab` option needs a directory after it where it can write the files. The SQL file is owned by whichever user we used to run the `mysqldump` command. The TXT file, on the other hand is owned by the `mysql` user, so whatever directory we specify needs to have permissions so that both users can write to it. The `/tmp/` directory is used in the example since, by default on Linux, this directory can be written to by anyone.

So why would a tab-delimited file of our data be useful? Well, for starters, the `mysqlimport` program reads tab-delimited files. Popular spreadsheets also read and write to tab-delimited files. So, for example, if we've been keeping our data in a spreadsheet, and have decided to move it to a MariaDB database, we can export our spreadsheet data as a tab-delimited file, create the tables in MariaDB, and then use `mysqlimport` to import our data. At a later point, we could use `mysqldump` to dump the data or a subset of the data to a file and then open it with our spreadsheet program, and create some nice pie charts or other graphs.

There are scores of other options that we can use to tweak and customize what and how `mysqldump` backs up our data. It's well worth it to take the time necessary to learn all of its many options.

Full documentation of the `mysqldump` utility is found at <https://mariadb.com/kb/en/mysqldump/>.

Restoring and importing data with `mysqlimport`

We talked briefly about `mysqlimport` in the previous section. In short, this command is used to import data into MariaDB. This data could be a backup that we made previously or completely new data. There are several options, but the basic syntax is as follows:

```
mysqlimport [--local] [-u username] [-p] database_name filename
```

The `filename` attribute must be the name of the table we want to import into. The `--local` option tells `mysqlimport` to read from the local filesystem instead of from the data directory of the server.

Here is an example importing the `employees.txt` file that we generated earlier:

```
sudo mysqlimport --local -u root -p test /tmp/employees.txt
```

Any records that cannot be imported will be skipped and not imported, and `mysqlimport` will report this and generate a warning. An example would be if our file has a column in it that contains values that must be unique in our database and if some of them match the existing records in the database.

Full documentation of the `mysqlimport` utility is found at <https://mariadb.com/kb/en/mysqlimport>.

Making backups of MyISAM tables quickly with `mysqlhotcopy`

The `mysqlhotcopy` backup program is actually a Perl script. It can make backups quickly, but only if our tables use the **MyISAM** or **ARCHIVE** storage engines. The default storage engine for MariaDB is **InnoDB**, so this script is less useful than it used to be back when **MyISAM** was the default storage engine. If we do have the **MyISAM** tables, however, it remains a useful tool.

The basic syntax of the `mysqlhotcopy` command is as follows:

```
mysqlhotcopy db_name [/path/to/new_directory]
```

If a path to a new directory is not given, `mysqlhotcopy` will make the backup in the MariaDB data directory.

Other limitations are that the command must be run by a user that can read the data files in the data directory, and when connecting to MariaDB, if we use a password, we must specify it on the command line or specify it in a `my.cnf` file as `mysqlhotcopy` does not support prompting us for the password.

Full documentation of the `mysqlhotcopy` program is found at <https://mariadb.com/kb/en/mysqlhotcopy/>.

Making cold backups

Another option for backing up MariaDB is to just copy the entire data directory. This is called a **cold backup**. As mentioned at the beginning of this section, problems can arise if we try to do this while MariaDB is running, but if we stop MariaDB briefly, and are using a filesystem that supports **snapshots** (called **shadow volume copies** on Windows) we can stop MariaDB briefly, make a snapshot, and then start MariaDB. Total downtime for an operation such as this, depending on various factors, might be only a few seconds. The snapshotted directory may then be backed up like any other filesystem directory backup.

This is obviously not an ideal way to make backups in all situations, especially when stopping the database server, even for a few seconds, is not an option. But it can work very well in some cases.

Checking and repairing tables

After a hardware failure, after a power outage, or even after an upgrade, it is a good idea to check the tables in our MariaDB databases to make sure they are ok. MariaDB includes several utilities to do this.

Checking and optimizing tables with `mysqlcheck`

The `mysqlcheck` program can check, analyze, optimize, and repair MariaDB database tables. Basic syntax for the command is as follows:

```
mysqlcheck [options] [-u username] [-p] database_name [table_name]
```

Here is an example of running the command to check our test database, and its output:

```
daniel@gandalf:~$ mysqlcheck -u root -p test
Enter password:
test.employees                                OK
```

We can specify multiple databases using the `--databases` option as follows:

```
mysqlcheck -u root -p --databases db_name1 db_name2 db_name3
```

We can also tell the program to check all of our databases with the `--all-databases` option, as follows:

```
mysqlcheck -u root -p --all-databases
```

By default, `mysqlcheck` will only check tables when it is run. To get it to optimize, analyze, or repair tables, we use one of the following options:

- `--optimize`
- `--analyze`
- `--repair`

Not all of the options work on all tables. For example, InnoDB tables cannot be repaired with `mysqlcheck`. The program displays an error message if it cannot perform a requested action.

Full documentation of the `mysqlcheck` utility is found at <https://mariadb.com/kb/en/mysqlcheck/>.

Repairing tables

Thankfully, MariaDB is a very mature and stable program. Problems are few and very far between. However, power does sometimes go out and hardware sometimes fails catastrophically or gradually, so there may come a time when a table in our database has problems and needs to be repaired.

MyISAM and **Aria** tables can often be repaired with the `mysqlcheck` program, so if `mysqlcheck` reports that a table needs repairing then we can usually simply re-run the program with the `--repair` option. Unfortunately, `mysqlcheck` cannot repair InnoDB tables.

Thankfully, InnoDB and XtraDB are crash safe, which means they are protected to a certain extent when failures do occur. This protection means that the chances of a hardware failure causing corruption are very low. InnoDB and XtraDB also have a built-in crash-recovery mechanism. The way to use it is to add the `innodb_force_recovery` option to the `[mysqld]` section of our `my.cnf` or `my.ini` file with it set to a number between 1 and 6. Setting this variable to 0 or removing it entirely disables it. While this option is set, MariaDB will not allow any InnoDB tables to be changed. The higher the number, the more aggressively MariaDB will try to repair the tables. Full documentation of this feature is at <https://mariadb.com/kb/en/xtradbinnodb-recovery-modes/>.

If `innodb_force_recovery` does not work, we may need to dump and reload our affected tables. This procedure can take a long time on a large server, so it should only be used as a last resort. The basic procedure to dump and reload a database is as we went over in the `mysqldump` section previously, but here it is again:

```
mysqldump [options] database_name > dump.sql
mysql database_name < dump.sql
```

So to dump and reload our test database we might do the following:

```
mysqldump -u root -p test > dump.sql
mysql -u root -p test < dump.sql
```

This reload process can be more likely to succeed if it is used in conjunction with the `innodb_force_recovery` variable. For example, a setting of 1 tells InnoDB and XtraDB to skip corrupt indexes and records instead of attempting to read them. Refer to the **XtraDB/InnoDB Recovery Modes** page in the MariaDB Knowledge Base found previously in this section for more information.

If the preceding reload process doesn't fix the error, we might want to call in some experts. There are various recovery strategies out there, but they are beyond the scope of this book. We could also try reloading from a backup, we may lose some data depending on how old the backup is, but losing some data is better than losing everything.

Summary

In this chapter, we learned about the various MariaDB log files and what they are used for. We also went over common backup methods and programs. We briefly discussed optimization. Lastly, we wrapped things up with a discussion of various things that we can do if something goes wrong.

MariaDB Next Steps

This book provides an introduction to MariaDB, with enough information to get us started. MariaDB is a large system with many parts, options, and capabilities. So where do we go from here? If we have a question, where do we go for help? Here is a list of various online resources available to help us on our way to becoming a MariaDB expert.

Let's begin with the official MariaDB website. MariaDB downloads, the MariaDB Foundation blog, and other official MariaDB information is found at <http://mariadb.org>.

Next is the MariaDB Knowledge Base available at <https://mariadb.com/>. This is the official location of MariaDB documentation. New information is being added here on a daily basis. Whenever something is added to or changed in MariaDB it is documented here. Release notes and change logs for MariaDB releases are also posted here.

There is also an **Ask a Question** feature that can be used if you have a question about something in MariaDB. Just navigate to the section or item you are interested in, click on the button and ask away. You can also provide your own tips and tricks by leaving comments on the page. Registration is required (to cut down on spam) but it is free and all the content is released under either a Creative Commons, GFDL, or GPL license. The direct link to MariaDB Knowledge Base is <https://mariadb.com/kb/>.

If we want to talk to someone right now, there are a few options. First is IRC, where we can engage in real-time chat conversations with other users and with the developers of MariaDB. The official MariaDB channel is #maria on the Freenode IRC network. See the Knowledge Base entry on IRC (<https://mariadb.com/kb/en/irc>) for more information.

To engage in a direct conversation, there are three MariaDB e-mail lists. The developers list is for technical discussions about MariaDB development. The discuss list is for general discussions about using MariaDB. The docs list is for discussion and planning related to MariaDB documentation. All three lists are hosted on `launchpad.net`. The most useful list for end users is the discuss list. Following are the links to these lists:

- MariaDB developers list – <https://launchpad.net/~maria-developers>
- MariaDB discuss list – <https://launchpad.net/~maria-discuss>
- MariaDB docs list – <https://launchpad.net/~maria-docs>

Lastly, MariaDB is active on the major social media platforms. Here are the locations of the official MariaDB accounts on Twitter, Google+, and Facebook:

- Twitter – <http://twitter.com/mariadb>
- Google+ – <http://google.com/+mariadb>
- Facebook – <http://fb.com/MariaDB.dbms>

I hope you enjoy working with MariaDB!

Index

Symbols

--all-databases option 77
<alter_definition> attribute 60
<alter_definition> command 60
--databases option 76
--local option 75
-p option 52
--tab option 74
-u option 52
<what> section 65

A

Advanced Package Tool (APT) 22
Advanced Packaging Tool. *See* APT
AFTER option 60
Alpha 8
ALTER TABLE command 56, 60
APT 7
Aria table 77

B

backups
 creating, mysqldump used 73, 74
Beta 8
binary log, MariaDB
 about 69
 URL 70
 working 69, 70
Building 31

C

CentOS
 MariaDB, installing on 15, 16

cold backup
 creating 76
column
 adding 60
 dropping 61
 modifying 60
comments 25, 26
CREATE DATABASE command 54, 55
Create, Read, Update, and Delete (CRUD)
 68
CREATE TABLE command 56, 59, 62

D

data
 deleting 64
 importing, mysqlimport used 75
 inserting 62
 reading 64-67
 restoring, mysqlimport used 75
 updating 63, 64
database
 creating 54
 dropping 54, 55
 listing, on server 53
 using 52, 53
database and table privileges
 ALTER 42
 ALTER ROUTINE 42
 CREATE 42
 CREATE ROUTINE 42
 CREATE TEMPORARY TABLES 42
 CREATE VIEW 43
 DELETE 43
 DROP 43
 EVENT 43

- EXECUTE 43
- INDEX 43
- INSERT 43
- LOCK TABLES 43
- SELECT 43
- SHOW VIEW 43
- TRIGGER 43
- UPDATE 43
- datatype** 57
- Debian**
 - MariaDB, installing on 14
- DELETE command** 64
- delimiter** 53
- DESCRIBE command** 59
- development series** 8
- DROP command** 55
- DROP DATABASE command** 55
- DROP TABLE command** 61

E

- Enable networking option** 11
- error log, MariaDB**
 - about 70
 - URL 70
- event** 69

F

- Fedora**
 - MariaDB, installing on 15, 16
- filename attribute** 75
- filesystem layout, MariaDB**
 - about 21
 - comments 25- 27
 - configuration changes, activating 28
 - for Linux 22, 23
 - for Windows 21, 22
 - MariaDB configuration file 25
 - modular configuration 24
 - options 28
- FIRST option** 60
- foreign keys** 56

G

- general_log_file variable** 71
- general_log variable** 71

- general query log, MariaDB**
 - about 71
 - URL 71
- global administrative privileges**
 - about 42
 - CREATE USER 42
 - FILE 42
 - PROCESS 42
 - RELOAD 42
 - REPLICATION CLIENT 42
 - REPLICATION SLAVE 42
 - SHOW DATABASES 42
 - SHUTDOWN 42
 - SUPER 42
- Gnu Privacy Guard (GPG)** 18

I

- IF NOT EXISTS command** 57
- InnoDB** 75
- innodb_force_recovery variable** 78
- innodb variable** 72
- INSERT command** 63
- Install as service box** 11
- installing**
 - MariaDB, on CentOS 15, 17
 - MariaDB, on Debian 14, 15
 - MariaDB, on Fedora 15-17
 - MariaDB, on Linux Mint 14, 15
 - MariaDB, on Mac OS X 13
 - MariaDB, on other Linux distribution 17
 - MariaDB, on Red Hat 15-17
 - MariaDB, on Ubuntu 14, 15
 - MariaDB, on Windows 10-13
- Internal Network** 31
- internal network security** 38
- Internet** 31
- internet security** 39

L

- Linux**
 - filesystem layout 22, 23
- Linux Mint**
 - MariaDB, installing on 14
- log_bin variable** 69
- log_error variable** 70
- log_slow_rate_limit variable** 72

log_slow_verbosity variable 72
long_query_time variable 71

M

Mac OS X

MariaDB, installing on 13

maintenance series 9

MariaDB

about 21, 51, 79
backing up 73-76
comparison operators, URL 66
connecting to 34-36
database, creating 54
database, dropping 54, 55
database, using 52, 53
data, deleting 64
data, inserting 62
data, reading 64-67
data, updating 63, 64
end users list 80
filesystem layout 21
installing, on CentOS 15-17
installing, on Debian 14, 15
installing, on Fedora 15-17
installing, on Linux Mint 14, 15
installing, on Mac OS X 13
installing, on other Linux Distributions 17
installing, on Red Hat 15-17
installing, on Ubuntu 14, 15
installing, on Windows 10-13
internal network security 38
internet security 39
Knowledge Base 79
log files 69-72
on Facebook 80
on Google+ 80
on Twitter 80
optimizing 72, 73
optimizing, URL 73
post installation 18, 19
securing 32, 34
security 31
security, building 37
server security 36
supported data types, URL 57
tuning 72, 73

tuning, URL 73
user privileges 41
Window installations, MSI packages 10
Window installations, ZIP files 10
working 52

MariaDB back up

basic backups, mysqldump used 73, 74
cold backups, creating 76
data importing, mysqlimport used 75
data restoring, mysqlimport used 75
MyISAM tables backup, creating 75

MariaDB configuration file 25

MariaDB installation

issues 19

MariaDB log files

about 69
binary log 69, 70
error log 70
general query log 71
slow query log 71, 72
types 69

MariaDB Package Security 18

MariaDB series

choosing 8
development series 8
maintenance series 9
stable series 9

microtime variable 72

miscellaneous privileges

about 43
ALL PRIVILEGES 43
GRANT OPTION 43
limits, MAX_CONNECTIONS_PER_HOUR 44
limits, MAX_QUERIES_PER_HOUR 44
limits, MAX_UPDATES_PER_HOUR 44
limits, MAX_USER_CONNECTIONS 44
USAGE 43

modular configuration 23, 24

my.cnf files function 23

MyISAM table 77

MyISAM tables backup

creating, mysqlhotcopy used 75

mysqlbinlog program 70

mysqlcheck program

URL 77
used, for checking tables 76, 77

- used, for optimizing tables 76, 77
- used, for repairing tables 77, 78
- mysql command-line client**
 - running 51, 52
- mysqldump**
 - URL 74
 - used, for backup creating 73, 74
- mysqlhotcopy**
 - URL 76
 - used, for MyISAM tables backup creating 75
- mysqlimport**
 - used, for data importing 75
 - used, for data restoring 75

N

normalization 56

O

Optimize for transactions checkbox 12
ORDER BY clause 66

P

passwords

- changing 48

performance_schema database 54

permissions

- granting 45
- grants, showing 47

physical security 32

Pretty Good Privacy (PGP) 18

primary key 58

R

Red Hat

- MariaDB, installing on 15, 16

relational database 56

S

SELECT command 64, 67

series 8

server

- about 32
- databases, listing on 53

server security, MariaDB 36

shadow volume copies. *See* snapshots

SHOW CREATE TABLE command 58

SHOW DATABASES command 53

SHOW GRANTS command 48

slow_query_log_file variable 71

slow query log, MariaDB

- about 71
- log_slow_rate_limit variable 72
- log_slow_verbosity variable 72
- long_query_time variable 71
- slow_query_log_file variable 71
- switching on 72
- URL 72

snapshots 76

SQL 53

stable series 9

Structured Query Language. *See* SQL

T

table alteration

- column, adding 60
- column, dropping 61
- column, modifying 60

table creation

- create table command, showing 58, 59
- table structure, exploring 59

tables

- altering 60, 61
- checking, mysqlcheck used 76, 77
- creating 56-59
- dropping 61
- optimizing, mysqlcheck used 76, 77
- repairing, mysqlcheck used 77, 78

test database 58

U

Ubuntu

- MariaDB, installing on 14

UPDATE command 63, 64

USE command 52

user privileges

- adding 47
- database and table privileges 42
- global administrative privileges 42
- miscellaneous privileges 43

- removing 47
- users**
 - creating 44
 - removing 48

W

- WHERE clause** 67
- wildcard character** 67
- Windows**
 - filesystem layout 21, 22
 - MariaDB, installing on 10-12

X

- XtraDB/InnoDB Recovery Modes page** 78

Y

- Yellow Dog Updater, Modified.** *See* **YUM**
- Yellow Dog Updater, Modified (YUM)** 22
- YUM** 7



Thank you for buying **Getting Started with MariaDB**

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: www.packtpub.com.

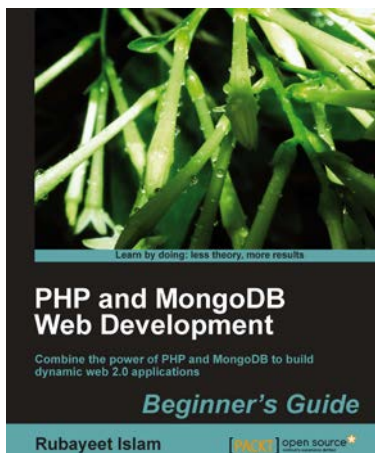
About Packt Open Source

In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt Open Source brand, home to books published on software built around Open Source licences, and offering information to anybody from advanced developers to budding web designers. The Open Source brand also runs Packt's Open Source Royalty Scheme, by which Packt gives a royalty to each Open Source project about whose software a book is sold.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.



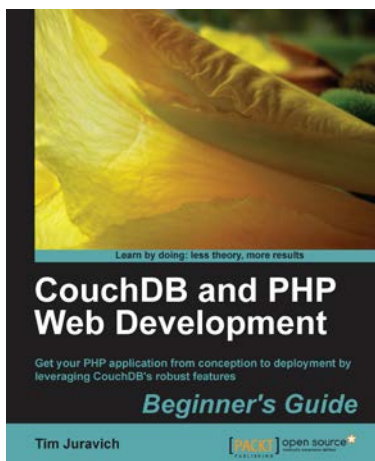
PHP and MongoDB Web Development Beginner's Guide

ISBN: 978-1-84951-362-3

Paperback: 292 pages

Combine the power of PHP and MongoDB to build dynamic web 2.0 applications

1. Learn to build PHP-powered dynamic web applications using MongoDB as the data backend
2. Handle user sessions, store real-time site analytics, build location-aware web apps, and much more, all using MongoDB and PHP
3. Full of step-by-step instructions and practical examples, along with challenges to test and improve your knowledge



CouchDB and PHP Web Development Beginner's Guide

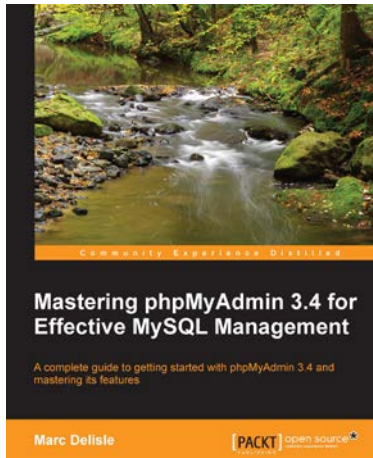
ISBN: 978-1-84951-358-6

Paperback: 304 pages

Get your PHP application from conception to deployment by leveraging CouchDB's robust features

1. Build and deploy a flexible Social Networking application using PHP and leveraging key features of CouchDB to do the heavy lifting
2. Explore the features and functionality of CouchDB, by taking a deep look into Documents, Views, Replication, and much more.
3. Conceptualize a lightweight PHP framework from scratch and write code that can easily port to other frameworks

Please check www.PacktPub.com for information on our titles

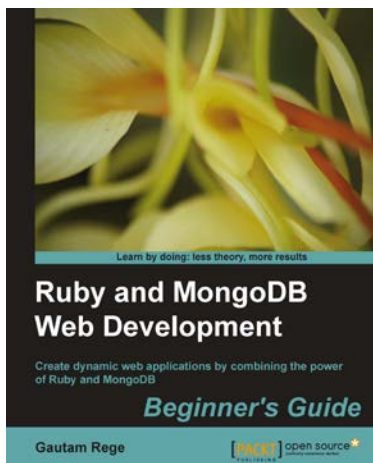


Mastering phpMyAdmin 3.4 for Effective MySQL Management

ISBN: 978-1-84951-778-2 Paperback: 394 pages

A complete guide to getting started with phpMyAdmin 3.4 and mastering its features

1. A step-by-step tutorial for manipulating data with the latest version of phpmyadmin
2. Administer your MySQL databases with phpMyAdmin
3. Manage users and privileges with MySQL Server Administration tools
4. Learn to do things with your MySQL database and phpMyAdmin that you didn't know were possible!



Ruby and MongoDB Web Development Beginner's Guide

ISBN: 978-1-84951-502-3 Paperback: 332 pages

Create dynamic web applications by combining the power of Ruby and MongoDB

1. Step-by-step instructions and practical examples to creating web applications with Ruby and MongoDB
2. Learn to design the object model in a NoSQL way
3. Create objects in Ruby and map them to MongoDB

Please check www.PacktPub.com for information on our titles