Getting Started with MariaDB
Second Edition

MariaDB is a database that has become very popular in the few short years that it has been around. It does not require a big server or expensive support contract. It is also powerful enough to be the database of choice for some of the biggest and most popular websites in the world, taking full advantage of the latest computing hardware available.

From installing and configuring through basic usage and maintenance, each chapter in this revised and expanded guide leads on sequentially and logically from the one before it, introducing topics in their natural order so you learn what you need, when you need it. The book is based on the latest release of MariaDB and covers all the latest features and functions. By the end of this beginner-friendly book, not only will you have a running installation of MariaDB, but you will have practical, hands-on experience in the basics of how to install, configure, administer, use, and maintain it.

Who this book is written for
This book is for anyone who wants to learn more about databases in general and wants to get started with MariaDB. Prior database experience is not required.

What you will learn from this book
- Install MariaDB on Windows, Mac OS X, and Linux
- Configure MariaDB for better performance using the features of version 10
- Get familiar with the usage of operators for retrieving rows more selectively
- Use MariaDB 10’s powerful tools to store, retrieve, and analyze data
- Secure MariaDB from unauthorized access
- Maintain the databases associated with MariaDB to ensure efficient and optimum functioning
- Retrieve data and learn how to sort, summarize, group, and manipulate it.

Who this book is written for
This book is for anyone who wants to learn more about databases in general and wants to get started with MariaDB. Prior database experience is not required.

Get familiar with the usage of operators for retrieving rows more selectively.
Use MariaDB 10’s powerful tools to store, retrieve, and analyze data.
Secure MariaDB from unauthorized access.
Maintain the databases associated with MariaDB to ensure efficient and optimum functioning.
Retrieve data and learn how to sort, summarize, group, and manipulate it.

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 5 'Using MariaDB – Databases and Tables'
- A synopsis of the book’s content
- More information on Getting Started with MariaDB Second Edition
About the Author

Daniel Bartholomew has been using Linux since 1997 and databases since 1998. In addition to this book, he has also written MariaDB Cookbook, Packt Publishing, and dozens of articles for various magazines, including The Linux Journal, Linux Pro, Ubuntu User, and Tux. He became involved with the MariaDB project shortly after it began in early 2009 and continues to be involved to this day. He currently works for MariaDB, Inc. and splits his time between managing MariaDB releases, documentation, and maintaining various bits and pieces that keep the MariaDB project running smoothly.
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, hospitals, amusement parks, and police departments. All use databases to store, sort, and analyze their own particular 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 much more. Databases are one of the foundational pillars of the modern electronic world.

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

There is a new database that has caught the attention of the database community over the past few years like few others have. First released in 2009, its name is MariaDB—named after the youngest daughter of its creator, Michael "Monty" Widenius.

MariaDB may be younger than the databases it is often compared with, but it has a stellar parentage. It's a next-generation evolution 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 that 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 faster 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 MariaDB is a very powerful and capable database, make no mistake—it is very easy to install and use.

This book provides an introduction to MariaDB that is enough to get us started. Don't worry if you've never used a database before - this book covers everything you need to know, and before you know it, you'll be on your way to becoming an expert database administrator (DBA). But even if you never move beyond just tinkering or playing around with MariaDB, you'll learn 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, Linux, and Mac OS X.

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, Securing MariaDB, provides an overview of the best practices for MariaDB security, including how to easily secure a new MariaDB installation.

Chapter 4, Administering MariaDB, explains how to add and administer MariaDB user accounts.

Chapter 5, Using MariaDB – Databases and Tables, covers the commands used to create, update, and delete databases and tables.

Chapter 6, Using MariaDB – Inserting, Updating, and Deleting, covers the commands used to add, update, and delete data from our database tables.

Chapter 7, Using MariaDB – Retrieving Data, covers the commands used to retrieve data from our database tables, including filtering, searching, sorting, joining, and summarizing the data.
Chapter 8, Maintaining MariaDB, explains how to maintain your MariaDB database and keep it running smoothly.

Appendix, MariaDB Next Steps, provides you with a list of various online resources available to help you on your way to becoming a MariaDB expert.
From this chapter onwards, we will focus on using the command-line `mysql` client to perform common tasks. In this chapter, you'll learn about the following:

- The `mysql` command-line client application
- Connecting to MariaDB
- Using `USE` to select a database
- Using `SHOW` to list all databases on a server
- Creating and deleting databases
- Data, tables, and normalization
- Creating, altering, and dropping tables

**The `mysql` command-line client application**

A big part of becoming a MariaDB expert is learning how to effectively and efficiently use the command-line `mysql` client program. Often we will interact with MariaDB using custom programs that have been developed for specific needs. At a lower level though, every interaction that these programs 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 a client, which is the part that 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.

**Connecting to MariaDB**

To start the client and connect to MariaDB, 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 <username>] [-p] [-h <host>] [<database>]
```

All the options in the previous syntax example are in square 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 username (-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.

It is possible to add the password after `-p` on the command line, with a couple of caveats. First, there can't be a space between the `-p` and the password. For example if our username is tom and our password is `correcthorse` we can use the following command line to log in to MariaDB:

```
mysql -u tom -pcorrecthorse
```

The second caveat is that doing this is very insecure and should not, in fact, be done. Command-line interpreters and shells are almost always configured to save the commands we run in a history file that could have insecure permissions, meaning that if we make a habit of typing out our password on the command line like this, an attacker only has to gain access to the history file to find out our MariaDB user password.

A successful connection will look similar to the following:

```
daniel@pippin:-$ mysql -u root -p
Enter password:
Welcome to the MariaDB monitor. Commands end with ; or \g.
```
Chapter 5

Your MariaDB connection id is 209
Server version: 10.1.2-MariaDB-1-trusty-wsrep-log mariadb.org binary
distribution, wsrep_25.10.r4123

Copyright (c) 2000, 2014, Oracle, SkySQL 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 the brackets tells us which database on the server we are currently using by default; in this case, we aren't using any database, so it says (none).

Using USE to select 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;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [test]>

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

MariaDB [(none)]> USE test1;
ERROR 1049 (42000): Unknown database 'test1'
Using SHOW to list all databases on a server

To show a list of all of the databases on a server that the current user is allowed to see, use the `SHOW DATABASES` command as in the following example:

```
MariaDB [(none)]> SHOW DATABASES;
+-------------------+
| Database           |
+-------------------+
| dbt3_s001         |
| flightstats       |
| ham               |
| information_schema|
| isfdb             |
| lds_scriptures    |
| library           |
| mysql             |
| performance_schema|
| test              |
| wikidb            |
+-------------------+
11 rows in set (0.00 sec)
```

MariaDB [(none)]>

The preceding example is from my personal install of MariaDB; the databases listed when you run the command will almost assuredly be different. This command is useful especially if you're given access to an existing MariaDB database server and want to see what databases are available to you, or if you can't quite remember what a specific database was named.
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 can 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 the 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 database servers that use their own variant of SQL.

Creating and deleting 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, databases, and other housekeeping information. The installer also creates a test database for experimentation and learning, and a couple of read-only, semi-virtual databases where MariaDB stores performance and other statistics. We don't want to use the system database as this could mess up the entire server if we made a mistake. We can't put data into the statistics databases, called information_schema and performance_schema, because they are semi-virtual and 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.

Another word for a database is schema. In some database servers, a schema and a database are not quite the same thing, but in MariaDB they are. So when we see information_schema, this means the information database. We can even use SCHEMA instead of DATABASE when we are using the command-line client. For example: SHOW SCHEMAS instead of SHOW DATABASES. In this book, we'll stick to the name databases.
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.

**Using CREATE DATABASE to create a database**

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

```
CREATE DATABASE <databasename>;
```

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`.

The following are some examples:

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

The preceding 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 the following location:


**Using DROP DATABASE to delete 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 DATABASE` command. Out of the database commands, this one is by far the easiest, but it is potentially the most dangerous. 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`.

The following are a couple of examples that drop the database that 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.

As mentioned previously, the `DROP DATABASE` command can be very dangerous. Why is this, you might ask? This is because if you have the appropriate permission to drop a database, MariaDB trusts you and will delete the database and everything in it when you tell it to, no questions asked. So when setting up users, it is important to give only trusted users, who actually need it, the ability to use the `DROP DATABASE` command. More on setting up users and giving them permissions is given in Chapter 4, *Administering MariaDB*.

Warning: 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, *Administering MariaDB*, for information on managing users and their privileges.

Complete documentation of the `DROP DATABASE` command is available at the following location:


### Data, tables, and normalization

The primary purpose of a database is to store data. Data is information, usually text-based, but not always, and this data could be anything from a company phone directory, to patient medical information, to an auto parts list, or even reviews of gourmet hot sauces complete with pictures of the bottles.

Database servers such as MariaDB store information, no matter what it is, in a structure called a *table*. 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 columns in a database can have relationships defined 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 server.

A database without tables of data is nothing more than an entry in the MariaDB system database (this database is called mysql) and a directory in the file system under the datadir directory. Until we create some tables and start adding data to those tables, our new database is useless.

There are few things in MariaDB that we will spend more time on, at least in the beginning, than when we create the tables in our database.

When we create a table, we are defining its structure. This structure includes such things as the number of columns and the type of data that we want to store in each column. Data types include things such as numbers, text, and dates. For example, if we are creating an employee table, we might decide that each row will contain an employee identification number (number), a surname (text), any given names (text), a preferred name (text), a 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 necessarily want to store that kind of data in the same table. Why? Because they are things people often have more than one of. For example, many people have both personal and work e-mail addresses. The same holds true for phone numbers and, for some people, even houses. If we try to design a table that has enough fields for the multiples of phone numbers and e-mail addresses that people have, it will quickly become unwieldy with too many columns, and with possibly no single row that uses all of them. Instead, we break apart the data into multiple tables, and define the relationships between the tables.

A good rule of thumb is to break the information apart into a separate table when it is clear there could be multiples of it. For example, it wouldn't make sense to have a single orders table in a company database that contains everything. Instead, we would have a customers table for the core customer information, an addresses table to hold the multiple addresses that the customers might want us to ship items to, an items table for the various things we might ship to a customer, and lastly, an orders table to actually track the orders made by customers. Of course, this is only one way to split the information apart and we might also need to store payment, supplier, and other information.
The process by which we refine our table definitions and split our data off into multiple tables 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 and you can refer to the following location:


Creating, altering, and dropping tables
Now that we know a little about how data is structured in a database, we can learn more about creating our own tables, making changes to them, and even how to delete them.

Using CREATE TABLE
We use the CREATE TABLE command to create tables. 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 about this too much, as we can always make changes later with the ALTER TABLE command (see the Using ALTER TABLE section later in this chapter).

Using CREATE TABLE – basic syntax
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 syntax:

<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 undefined; some value has to be assigned to it, even if the value assigned is an empty value. Multiple column definitions are separated by commas.

Using CREATE TABLE – datatypes

There are many different datatypes (given by <data_type> in the column definitions syntax example shown previously) to choose from. A datatype is the type of data being stored. Different datatypes exist because various types of data are most efficiently stored in different ways. Plain numbers can be treated differently than dates and vice versa. Common datatypes include numeric (numbers), strings (text), and dates.

Numeric datatypes include INTEGER (basic numbers, commonly written as INT), and FLOAT (for floating point numbers). A good article on floating point numbers can be found at the following location:

http://en.wikipedia.org/wiki/Floating_point

String (or text-based) datatypes include CHAR, TEXT, and VARCHAR, which are optimized for different lengths of text. The CHAR datatype is for fixed-length strings, for example, a product identifier that contains both numbers and letters can't be stored as a number, but if it is a fixed length such as 8 characters, we can store it efficiently as CHAR(8).

The VARCHAR datatype is for text that isn't more than a sentence or so long. Text such as names and addresses are commonly stored as VARCHAR.

Lastly, date and time datatypes include DATE, TIME, and DATETIME. As you might guess, the DATE datatype is for storing dates. Dates are always stored and displayed in the form YYYY-MM-DD (a four digit year, a two digit month, and a two digit day), for example 1998-02-14, and while it is recommended to input them that way, they can be entered in a variety of ways. For example: 2015-5-28, 15528, and 15*05*28 are all ways to enter the date 2015-05-28.

The TIME datatype is for time in the format HH:MM:SS.ssssss (hours:minutes:seconds.microseconds). As with the DATE datatype, while MariaDB will store and display values in those formats, it is less picky about how they are entered.
The `DATETIME` datatype is a combination of both the `DATE` and `TIME` datatypes. It stores and displays values in the following form: `YYYY-MM-DD HH:MM:SS.ssssss` and unlike the `TIME` datatype, the hours, minutes, and seconds should conform to real-world values (no 26 hour days, for example).

There are other specialized datatypes that can be used with MariaDB, but these are enough to get us started. See a complete list of supported datatypes at the following location:


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

**Using CREATE TABLE – other options**

After specifying the type, length, and precision (which are only required for some datatypes), we can specify other options. We can specify whether or not a column is allowed to be undefined (or `NULL`), what the default value (`<default_value>` in the syntax example) is, if anything, whether the column auto-increments (only for numeric datatypes such as `INT` and `FLOAT`), whether the value in the column should be `UNIQUE` (meaning whether or not it is allowed to have the same value as the same column in a different row), whether the column is a primary key, and a comment describing the table, if desired.

A primary key is a column, or a 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.

**Using CREATE TABLE – an example**

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):

```sql
CREATE TABLE employees (  
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,  
    surname VARCHAR(100),  
    givenname VARCHAR(100),  
    pref_name VARCHAR(50),  
    birthday DATE COMMENT 'approximate birthday OK'
);```
When we run the preceding code, the output looks as follows:

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 is a brand new table and thus has no data in it yet. Unless we are on a very slow, or a very busy server, the command should complete instantly (0.00 seconds) or near instantly (such as 0.05 seconds).

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

Using SHOW to display 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. Take a look at the following 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 COMMENT 'approximate birthday is OK',
  PRIMARY KEY (`id`) ) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)

The \G at the end of the first line 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 settings.

For example, the `ENGINE` and `DEFAULT CHARSET` parts after the column definitions are default table options on my local MariaDB server. 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 the following location:

### Using DESCRIBE to explore 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:

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

```
+-----------+--------------+------+-----+---------+----------------+
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Key</th>
<th>Default</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>int(11)</td>
<td>NO</td>
<td>PRI</td>
<td>NULL</td>
<td>auto_increment</td>
</tr>
<tr>
<td>surname</td>
<td>varchar(100)</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>givenname</td>
<td>varchar(100)</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>pref_name</td>
<td>varchar(50)</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>birthday</td>
<td>date</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
</tbody>
</table>
+-----------+--------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

This basic information comes in handy especially when we want to look up information in a table that we are unfamiliar with, or if we can't remember all the fields. (Looking up information is covered in *Chapter 7, Using MariaDB – Selecting, Sorting, and Searching*).

Another thing to note about the `DESCRIBE` command is that `COMMENT` is not displayed.
If we are just interested in a specific column, we can specify it as follows:

```sql
MariaDB [test]> DESCRIBE employees birthday;
+----------+------+------+-----+---------+-------+
| Field    | Type | Null | Key | Default | Extra |
+----------+------+------+-----+---------+-------+
| birthday | date | YES  |     | NULL    |       |
+----------+------+------+-----+---------+-------+
1 row in set (0.00 sec)
```

Full documentation of the `DESCRIBE` command can be found at the following location:

https://mariadb.com/kb/en/describe/

---

Using **ALTER TABLE**

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

**Using ALTER TABLE – basic syntax**

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

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

The `<alter_definition>` part of the 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 show actual usage will contain the full command.

When using the `ALTER TABLE` command, the data in our table is preserved and converted when necessary.
Using ALTER TABLE – adding a column

An alter definition of an ALTER TABLE command 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 a row. The AFTER option lets us specify which column the new column appears after. If we don't use FIRST or AFTER, the column will be added after the current last column. 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;

Using ALTER TABLE – 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) from its original setting of varchar(50):

ALTER TABLE employees MODIFY pref_name varchar(25);

Using ALTER TABLE – dropping a column

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

DROP <column_name>

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

ALTER TABLE employees DROP username;
If you've been following along with these `ALTER TABLE` commands, your `employees` table should now look as follows:

```sql
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(25)  | YES  |     | NULL    |                |
| birthday  | date         | YES  |     | NULL    |                |
+-----------+--------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

Full documentation of the `ALTER TABLE` command is found at the following location:


### Using DROP TABLE

When we no longer need a table, just as when we no longer need a database, we use the `DROP TABLE` command to delete it. Out of the table commands, this one is by far the easiest, but it is potentially the most dangerous. The basic syntax of the command is as follows:

`DROP TABLE <table_name>`

If we try issuing a `DROP TABLE` on a table that doesn't exist, we will receive an error. We can suppress the error with `IF EXISTS`. The following are a couple of examples of this:

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

If the table exists, the preceding two commands have the same result, the `mytable` table is deleted. If the table doesn't exist, the first command will exit with an error and the second command will do nothing.
As mentioned previously, the `DROP TABLE` command can be very dangerous, because if you have the appropriate permission to drop a table, MariaDB trusts you and will delete the table and everything in it when you tell it to, no questions asked. So when setting up users, it is important to give only a few trusted users, who actually need it, the ability to use the `DROP TABLE` command. For more on setting up users and giving them permissions, refer Chapter 4, Administering MariaDB.

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

Summary
In this chapter, we covered a lot of ground. We learned about the `mysql` command-line client application and how to use it to connect to MariaDB, how to use the `USE` command to switch to an existing database, how to use the `SHOW` command to list all the databases in MariaDB, and how to create our own databases. We also explored a little of the information regarding datatypes and normalization, and learned how to use that information in order to create and modify our own tables. Lastly, we learned how to delete (`DROP`) both, databases and tables. In the next chapter, we'll start storing and modifying the data in the tables of our database.
Where to buy this book

You can buy Getting Started with MariaDB Second Edition from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.