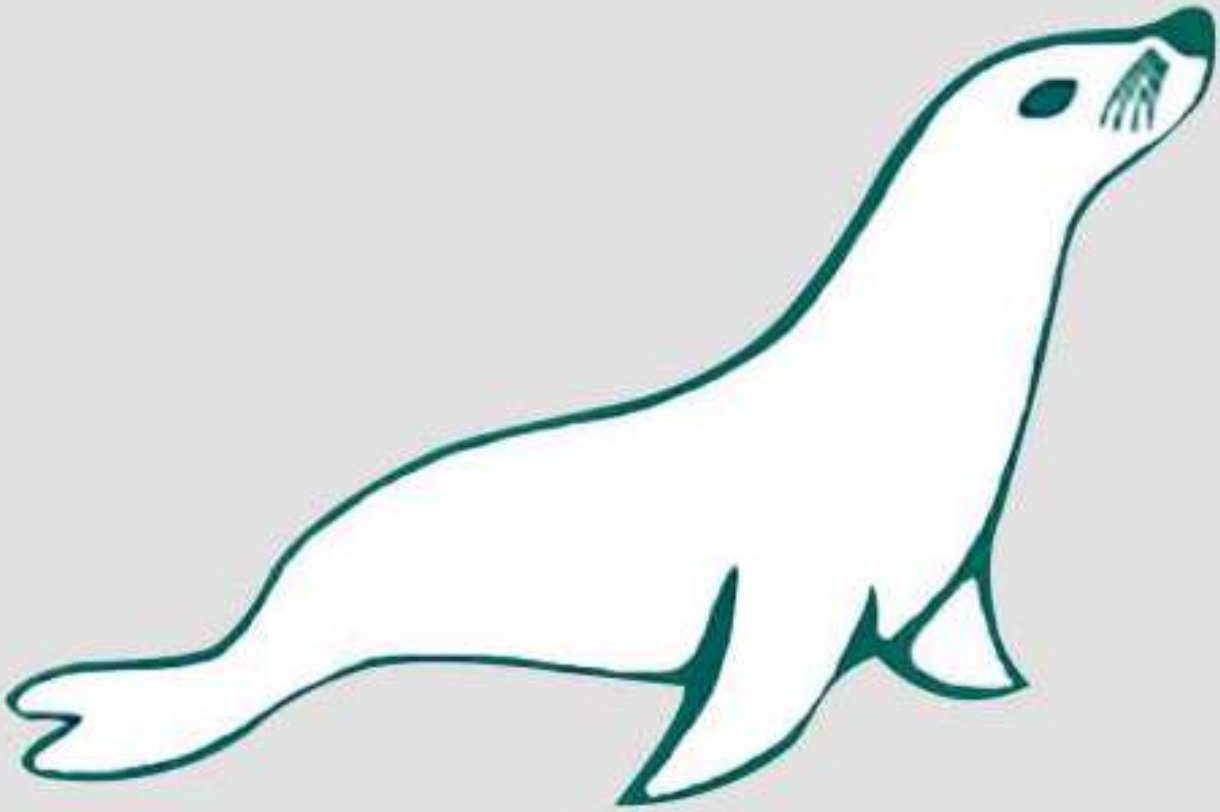

MariaDB

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia https://twitter.com/tutorialspoint

About the Tutorial

MariaDB is a fork of the MySQL relational database management system. The original developers of MySQL created MariaDB after concerns raised by Oracle's acquisition of MySQL.

This tutorial will provide a quick introduction to MariaDB and aid you in achieving a high level of comfort with MariaDB programming and administration.

Audience

This tutorial targets novice developers and those new to MariaDB. It guides them in understanding basic through more advanced concepts in MariaDB. After completing this tutorial, your firm foundation in MariaDB and level of expertise will allow you to begin developing and easily build on your knowledge.

Prerequisites

The tutorial assumes that you are familiar with relational database management systems, querying languages, MySQL, and general programming. It also assumes familiarity with typical database operations in an application.

Copyright & Disclaimer

Table of Contents

1. MariaDB – Introduction

A database application exists separate from the main application and stores data collections. Every database employs one or multiple APIs for the creation, access, management, search, and replication of the data it contains.

Databases also use non-relational data sources such as objects or files. However, databases prove the best option for large datasets, which would suffer from slow retrieval and writing with other data sources.

Relational database management systems, or RDBMS, store data in various tables. Relationships between these tables are established using primary keys and foreign keys.

RDBMS offers the following features-

- They enable you to implement a data source with tables, columns, and indices.
- They ensure the integrity of references across rows of multiple tables.
- They automatically update indices.
- They interpret SQL queries and operations in manipulating or sourcing data from tables.

RDBMS Terminology

Before we begin our discussion of MariaDB, let us review a few terms related to databases.

- **Database** – A database is a data source consisting of tables holding related data.
- **Table** – A table, meaning a spreadsheet, is a matrix containing data.
- **Column** – A column, meaning data element, is a structure holding data of one type; for example, shipping dates.
- **Row** – A row is a structure grouping related data; for example, data for a customer. It is also known as a tuple, entry, or record.
- **Redundancy** – This term refers to storing data twice in order to accelerate the system.
- **Primary Key** – This refers to a unique, identifying value. This value cannot appear twice within a table, and there is only one row associated with it.
- **Foreign Key** – A foreign key serves as a link between two tables.
- **Compound Key** – A compound key, or composite key, is a key that refers to multiple columns. It refers to multiple columns due to a column lacking a unique quality.

- **Index –** An index is virtually identical to the index of a book.

- **Referential Integrity –** This term refers to ensuring all foreign key values point to existing rows.

MARIA Database

MariaDB is a popular fork of MySQL created by MySQL's original developers. It grew out of concerns related to MySQL's acquisition by Oracle. It offers support for both small data processing tasks and enterprise needs. It aims to be a drop-in replacement for MySQL requiring only a simple uninstall of MySQL and an install of MariaDB. MariaDB offers the same features of MySQL and much more.

Key Features of MariaDB

The important features of MariaDB are-

- All of MariaDB is under GPL, LGPL, or BSD.

- MariaDB includes a wide selection of storage engines, including high-performance storage engines, for working with other RDBMS data sources.

- MariaDB uses a standard and popular querying language.

- MariaDB runs on a number of operating systems and supports a wide variety of programming languages.

- MariaDB offers support for PHP, one of the most popular web development languages.

- MariaDB offers Galera cluster technology.

- MariaDB also offers many operations and commands unavailable in MySQL, and eliminates/replaces features impacting performance negatively.

Getting Started

Before you begin this tutorial, make sure you have some basic knowledge of PHP and HTML, specifically material discussed in our PHP and HTML tutorials.

This guide focuses on use of MariaDB in a PHP environment, so our examples will be most useful for PHP developers.

We strongly recommend reviewing our PHP Tutorial if you lack familiarity or need to review.

2. MariaDB – Installation

All downloads for MariaDB are located in the Download section of the official MariaDB foundation website. Click the link to the version you would like, and a list of downloads for multiple operating systems, architectures, and installation file types is displayed.

Installing on LINUX/UNIX

If you have intimate knowledge of Linux/Unix systems, simply download source to build your install. Our recommended way of installing is to utilize distribution packages. MariaDB offers packages for the following Linux/Unix distributions-

- RedHat/CentOS/Fedora
- Debian/Ubuntu

The following distributions include a MariaDB package in their repositories-

- openSUSE
- Arch Linux
- Mageia
- Mint
- Slackware

Follow these steps to install in an Ubuntu environment-

Step 1: Login as a root user.

Step 2: Navigate to the directory containing the MariaDB package.

Step 3: Import the GnuPG signing key with the following code-

```
sudo apt-key adv --recv-keys --keyserver keyserver.ubuntu.com 0xcbcb082a1bb943db
```

Step 4: Add MariaDB to the **sources.list** file. Open the file, and add the following code-

```
sudo add-apt-repository 'deb http://ftp.osuosl.org/pub/mariadb/repo/5.5/ubuntu
precise main'
```

Step 5: Refresh the system with the following-

```
sudo apt-get update
```

Step 6: Install MariaDB with the following-

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

Installing on Windows

After locating and downloading an automated install file (MSI), simply double click the file to start the installation. The installation wizard will walk you through every step of installation and any necessary settings.

Test the installation by starting it from the command prompt. Navigate to the location of the installation, typically in the directory, and type the following at the prompt-

```
mysqld.exe --console
```

If the installation is successful, you will see messages related to startup. If this does not appear, you may have permission issues. Ensure that your user account can access the application. Graphical clients are available for MariaDB administration in the Windows environment. If you find the command line uncomfortable or cumbersome, be sure to experiment with them.

Testing the Installation

Perform a few simple tasks to confirm the functioning and installation of MariaDB.

Use the Admin Utility to Get Server Status

View the server version with the mysqladmin binary.

```
[root@host]# mysqladmin --version
```

It should display the version, distribution, operating system, and architecture. If you do not see the output of that type, examine your installation for issues.

Execute Simple Commands with a Client

Bring up the command prompt for MariaDB. This should connect you to MariaDB and allow execution of commands. Enter a simple command as follows-

```
mysql> SHOW DATABASES;
```

Post- Installation

After successful installation of MariaDB, set a root password. A fresh install will have a blank password. Enter the following to set the new password-

```
mysqladmin -u root password "[enter your password here]";
```

Enter the following to connect to the server with your new credentials-

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

Upgrading on Windows

If you already have MySQL installed on your Windows system, and want to upgrade to MariaDB; do not uninstall MySQL and install MariaDB. This will cause a conflict with the existing database. You must instead install MariaDB, and then use the upgrade wizard in the Windows installation file.

The options of your MySQL my.cnf file should work with MariaDB. However, MariaDB has many features, which are not found in MySQL.

Consider the following conflicts in your my.cnf file-

- MariaDB uses Aria storage engine by default for temporary files. If you have a lot of temporary files, modify key buffer size if you do not use MyISAM tables.
- If your applications connect/disconnect frequently, alter the thread cache size.
- If you use over 100 connections, use the thread pool.

Compatibility

MySQL and MariaDB are essentially identical. However, there are enough differences to create issues in upgradation. Review more of these key differences in the MariaDB Knowledge Base.

3. MariaDB – Administration

Before attempting to run MariaDB, first determine its current state, running or shutdown. There are three options for starting and stopping MariaDB-

- Run mysqld (the MariaDB binary).
- Run the mysqld_safe startup script.
- Run the mysql.server startup script.

If you installed MariaDB in a non-standard location, you may have to edit location information in the script files. Stop MariaDB by simply adding a "stop" parameter with the script.

If you would like to start it automatically under Linux, add startup scripts to your **init** system. Each distribution has a different procedure. Refer to your system documentation.

Creating a User Account

Create a new user account with the following code-

```
 'newusername'@'localhost' IDENTIFIED BY 'userpassword';
```

This code adds a row to the user table with no privileges. You also have the option to use a hash value for the password. Grant the user privileges with the following code-

```
GRANT SELECT, INSERT, UPDATE, DELETE ON database1 TO 'newusername'@'localhost';
```

Other privileges include just about every command or operation possible in MariaDB. After creating a user, execute a "FLUSH PRIVILEGES" command in order to refresh grant tables. This allows the user account to be used.

The Configuration File

After a build on Unix/Linux, the configuration file "/etc/my.conf" should be edited to appear as follows-

```
# Example mysql config file.
# You can copy this to one of:
# /etc/my.cnf to set global options,
# /mysql-data-dir/my.cnf to get server specific options or
# ~/my.cnf for user specific options.
#
# One can use all long options that the program supports.
# Run the program with --help to get a list of available options
```

```
# This will be passed to all mysql clients

[client]
#password=my_password
#port=3306
#socket=/tmp/mysql.sock

# Here is entries for some specific programs
# The following values assume you have at least 32M ram

# The MySQL server
[mysqld]
#port=3306
#socket=/tmp/mysql.sock
temp-pool

# The following three entries caused mysqld 10.0.1-MariaDB (and possibly other
versions) to abort...
# skip-locking
# set-variable  = key_buffer=16M
# set-variable  = thread_cache=4

loose-innodb_data_file_path = ibdata1:1000M
loose-mutex-deadlock-detector
gdb

######### Fix the two following paths

# Where you want to have your database
data=/path/to/data/dir

# Where you have your mysql/MariaDB source + sql/share/english
language=/path/to/src/dir/sql/share/english

[mysqldump]
quick
set-variable = max_allowed_packet=16M
```

```
[mysql]
no-auto-rehash

[myisamchk]
set-variable= key_buffer=128M
```

Edit the lines "data= " and "language= " to match your environment.

After file modification, navigate to the source directory and execute the following-

```
./scripts/mysql_install_db    --srcdir=$PWD    --datadir=/path/to/data/dir    --
user=$LOGNAME
```

Omit the "$PWD" variable if you added datadir to the configuration file. Ensure "$LOGNAME" is used when running version 10.0.1 of MariaDB.

Administration Commands

Review the following list of important commands you will regularly use when working with MariaDB-

- **`USE [database name]`** – Sets the current default database.
- **`SHOW DATABASES`** – Lists the databases currently on the server.
- **`SHOW TABLES`** – Lists all non-temporary tables.
- **`SHOW COLUMNS FROM [table name]`** – Provides column information pertaining to the specified table.
- **`SHOW INDEX FROM TABLENAME [table name]`** – Provides table index information relating to the specified table.
- **`SHOW TABLE STATUS LIKE [table name]\G`** – Provides tables with information about non-temporary tables, and the pattern that appears after the LIKE clause is used to fetch table names.

4. MariaDB – PHP Syntax

MariaDB partners well with a wide variety of programming languages and frameworks such as PHP, C#, JavaScript, Ruby on Rails, Django, and more. PHP remains the most popular of all available languages due to its simplicity and historical footprint. This guide will focus on PHP partnered with MariaDB.

PHP provides a selection of functions for working with the MySQL database. These functions perform tasks like accessing it or performing operations, and they are fully compatible with MariaDB. Simply call these functions as you would call any other PHP function.

The PHP functions you will use for MariaDB conform to the following format-

```
mysql_function(value,value,...);
```

The second part of the function specifies its action. Two of the functions used in this guide are as follows-

```
mysqli_connect($connect);
mysqli_query($connect,"SQL statement");
```

The following example demonstrates the general syntax of a PHP call to a MariaDB function-

```
<html>
<head>
<title>PHP and MariaDB</title>
</head>
<body>
<?php
   $retval = mysql_function(value, [value,...]);
   if( !$retval )
   {
       die ( "Error: Error message here" );
   }
   // MariaDB  or PHP Statements
?>
</body>
</html>
```

In the next section, we will examine essential MariaDB tasks, using PHP functions.

5. MariaDB – Connection

One way to establish a connection with MariaDB consists of using the mysql binary at the command prompt.

MYSQL Binary

Review an example given below.

```
[root@host]# mysql -u root -p
Enter password:******
```

The code given above connects to MariaDB and provides a command prompt for executing SQL commands. After entering the code, a welcome message should appear indicating a successful connection, with the version number displayed.

```
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 122323232
Server version: 5.5.40-MariaDB-log

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

mysql>
```

The example uses root access, but any user with privileges can of course access the MariaDB prompt and perform operations.

Disconnect from MariaDB through the **exit** command as follows-

```
mysql> exit
```

PHP Connection Script

Another way to connect to and disconnect from MariaDB consists of employing a PHP script. PHP provides the **mysql_connect()** function for opening a database connection. It uses five optional parameters, and returns a MariaDB link identifier after a successful connection, or a false on unsuccessful connection. It also provides the **mysql_close()** function for closing database connections, which uses a single parameter.

Syntax

Review the following PHP connection script syntax-

```
connection mysql_connect(server,user,passwd,new_link,client_flag);
```

The description of the parameters is given below-

Parameter	Description
server	This optional parameter specifies the host name running the database server. Its default value is "localhost:.3036."
user	This optional parameter specifies the username accessing the database. Its default value is the owner of the server.
passwd	This optional parameter specifies the user's password. Its default value is blank.
new_link	This optional parameter specifies that on a second call to **mysql_connect()** with identical arguments, rather than a new connection, the identifier of the current connection will be returned.
client flags	This optional parameter uses a combination of the following constant values- • MYSQL_CLIENT_SSL – It uses ssl encryption. • MYSQL_CLIENT_COMPRESS – It uses compression protocol. • MYSQL_CLIENT_IGNORE_SPACE – It permits space after function names. • MYSQL_CLIENT_INTERACTIVE – It permits interactive timeout seconds of inactivity prior to closing the connection.

Review the PHP disconnection script syntax given below-

```
bool mysql_close (resource $link_identifier );
```

If you omit the resource, the most recent opened resource will close. It returns a value of true on a successful close, or false.

Try the following example code to connect with a MariaDB server-

```
<html>
<head>
<title>Connect to MariaDB Server</title>
</head>
<body>
<?php
   $dbhost = 'localhost:3036';
   $dbuser = 'guest1';
   $dbpass = 'guest1a';
   $conn = mysql_connect($dbhost, $dbuser, $dbpass);
   if(! $conn )
```

```
   {
     die('Could not connect: ' . mysql_error());
   }
   echo 'Connected successfully';
   mysql_close($conn);
?>
</body>
</html>
```

On successful connection, you will see the following output:

```
mysql> Connected successfully
```

6. MariaDB – Create Database

Creation or deletion of databases in MariaDB requires privileges typically only given to root users or admins. Under these accounts, you have two options for creating a database – the mysqladmin binary and a PHP script.

mysqladmin Binary

The following example demonstrates the use of the mysqladmin binary in creating a database with the name **Products**-

```
[root@host]# mysqladmin -u root -p create PRODUCTS
Enter password:******
```

PHP Create Database Script

PHP employs the **mysql_query** function in creating a MariaDB database. The function uses two parameters, one optional, and returns either a value of "true" when successful, or "false" when not.

Syntax

Review the following **create database script** syntax-

```
bool mysql_query (sql, connection );
```

The description of the parameters is given below-

Parameter	Description
sql	This required parameter consists of the SQL query needed to perform the operation.
connection	When not specified, this optional parameter uses the most recent connection used.

Try the following example code for creating a database-

```
<html>
<head>
<title>Create a MariaDB Database</title>
</head>
<body>
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully<br />';
$sql = 'CREATE DATABASE PRODUCTS';
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not create database: ' . mysql_error());
}
echo "Database PRODUCTS created successfully\n";
mysql_close($conn);
?>
</body>
</html>
```

On successful creation, you will see the following output:

```
mysql> Database PRODUCTS created successfully
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| PRODUCTS           |
+--------------------+
```

7. MariaDB – Drop Database

Creation or deletion of databases in MariaDB requires privileges, typically, only given to root users or admins. Under these accounts, you have two options for deleting a database: the mysqladmin binary and a PHP script.

Note that deleted databases are irrecoverable, so exercise care in performing this operation. Furthermore, PHP scripts used for deletion do **not** prompt you with a confirmation before the deletion.

mysqladmin Binary

The following example demonstrates how to use the mysqladmin binary to delete an existing database-

```
[root@host]# mysqladmin -u root -p drop PRODUCTS
Enter password:******
mysql> DROP PRODUCTS
ERROR 1008 (HY000): Can't drop database 'PRODUCTS'; database doesn't exist
```

PHP Drop Database Script

PHP employs the **mysql_query** function in deleting MariaDB databases. The function uses two parameters, one optional, and returns either a value of "true" when successful, or "false" when not. y

Syntax

Review the following drop database script syntax-

```
bool mysql_query( sql, connection );
```

The description of the parameters is given below-

Parameter	Description
sql	This required parameter consists of the SQL query needed to perform the operation.
connection	When not specified, this optional parameter uses the most recent connection used.

Try the following example code for deleting a database-

```
<html>
<head>
<title>Delete a MariaDB Database</title>
</head>
<body>
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully<br />';
$sql = 'DROP DATABASE PRODUCTS';
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not delete database: ' . mysql_error());
}
echo "Database PRODUCTS deleted successfully\n";
mysql_close($conn);
?>
</body>
</html>
```

On successful deletion, you will see the following output:

```
mysql> Database PRODUCTS deleted successfully
```

8. MariaDB – Select Database

After connecting to MariaDB, you must select a database to work with because many databases may exist. There are two ways to perform this task: from the command prompt or through a PHP script.

The Command Prompt

In choosing a database at the command prompt, simply utilize the SQL command '**use**'-

```
[root@host]# mysql -u root -p
Enter password:******
mysql> use PRODUCTS;
Database changed

mysql> SELECT database();

+--------------------+
| Database           |
+--------------------+
| PRODUCTS           |
+--------------------+
```

Once you select a database, all subsequent commands will operate on the chosen database.

Note : All names (e.g., database, table, fields) are case sensitive. Ensure commands conform to the proper case.

PHP Select Database Script

PHP provides the **mysql_select_db** function for database selection. The function uses two parameters, one optional, and returns a value of "true" on successful selection, or false on failure.

Syntax

Review the following select database script syntax.

```
bool mysql_select_db( db_name, connection );
```

The description of the parameters is given below-

Parameter	Description
db_name	This required parameter specifies the name of the database to use.
connection	When not specified, this optional parameter uses the most recent connection used.

Try the following example code for selecting a database-

```
<html>
<head>
<title>Select a MariaDB Database</title>
</head>
<body>
<?php
$dbhost = 'localhost:3036';
$dbuser = 'guest1';
$dbpass = 'guest1a';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
mysql_select_db( 'PRODUCTS' );
mysql_close($conn);
?>
</body>
</html>
```

On successful selection, you will see the following output:

```
mysql> Connected successfully
```

9. MariaDB – Data Types

Good field definitions are essential for the optimization of your database. The ideal approach requires that you exclusively use a field of the type and size needed. For example, if you will only use a field, five-characters wide, do not define a field, 20-characters wide. Field (or column) types are also known as data types given the data types stored within the field.

MariaDB data types can be categorized as numeric, date and time, and string values.

Numeric Data Types

The numeric data types supported by MariaDB are as follows-

- **TINYINT** – This data type represents small integers falling within the signed range of -128 to 127, and the unsigned range of 0 to 255.

- **BOOLEAN** – This data type associates a value 0 with "false," and a value 1 with "true."

- **SMALLINT** – This data type represents integers within the signed range of -32768 to 32768, and the unsigned range of 0 to 65535.

- **MEDIUMINT** – This data type represents integers in the signed range of -8388608 to 8388607, and the unsigned range of 0 to 16777215.

- **INT(also INTEGER)** – This data type represents an integer of normal size. When marked as unsigned, the range spans 0 to 4294967295. When signed (the default setting), the range spans -2147483648 to 2147483647. When a column is set to **ZEROFILL**(an unsigned state), all its values are prepended by zeros to place M digits in the INT value.

- **BIGINT** – This data type represents integers within the signed range of 9223372036854775808 to 9223372036854775807, and the unsigned range of 0 to 18446744073709551615.

- **DECIMAL**(also DEC, NUMERIC, FIXED) – This data type represents precise fixed-point numbers, with M specifying its digits and D specifying the digits after the decimal. The M value does not add "-" or the decimal point. If D is set to 0, no decimal or fraction part appears and the value will be rounded to the nearest DECIMAL on INSERT. The maximum permitted digits is 65, and the maximum for decimals is 30. Default value for M on omission is 10, and 0 for D on omission.

- **FLOAT** – This data type represents a small, floating-point number of the value 0 or a number within the following ranges-

    - -3.402823466E+38 to -1.175494351E-38
    - 1.175494351E-38 to 3.402823466E+38

- **DOUBLE** (also **REAL** and **DOUBLE PRECISION**)– This data type represents normal-size, floating-point numbers of the value 0 or within the following ranges-
  - -1.7976931348623157E+308 to -2.2250738585072014E-308
  - 2.2250738585072014E-308 to 1.7976931348623157E+308

- **BIT** – This data type represents bit fields with M specifying the number of bits per value. On omission of M, the default is 1. Bit values can be applied with " b'[value]' " in which *value* represents bit value in 0s and 1s. Zero-padding occurs automatically from the left for full length; for example, "10" becomes "0010."

Date and Time Data Types

The date and time data types supported by MariaDB are as follows-

- **DATE** – This data type represents a date range of "1000-01-01" to "9999-12-31," and uses the "YYYY-MM-DD" date format.

- **TIME** – This data type represents a time range of "-838:59:59.999999" to "838:59:59.999999."

- **DATETIME** – This data type represents the range "1000-01-01 00:00:00.000000" to "9999-12-31 23:59:59.999999." It uses the "YYYY-MM-DD HH:MM:SS" format.

- **TIMESTAMP** – This data type represents a timestamp of the "YYYY-MM-DD HH:MM:DD" format. It mainly finds use in detailing the time of database modifications, e.g., insertion or update.

- **YEAR** – This data type represents a year in 4-digit format. The four-digit format allows values in the range of 1901 to 2155, and 0000.

String DataTypes

The string type values supported by MariaDB are as follows-

- **String literals** – This data type represents character sequences enclosed by quotes.

- **CHAR** – This data type represents a right-padded, fixed-length string containing spaces of specified length. M represents column length of characters in a range of 0 to 255, its default value is 1.

- **VARCHAR** – This data type represents a variable-length string, with an M range (maximum column length) of 0 to 65535.

- **BINARY** – This data type represents binary byte strings, with M as the column length in bytes.

- **VARBINARY** – This data type represents binary byte strings of variable length, with M as column length.

- **TINYBLOB** – This data type represents a blob column with a maximum length of 255 (2^8 - 1) bytes. In storage, each uses a one-byte length prefix indicating the byte quantity in the value.

- **BLOB** – This data type represents a blob column with a maximum length of 65,535 (2^{16} - 1) bytes. In storage, each uses a two-byte length prefix indicating the byte quantity in the value.

- **MEDIUMBLOB** – This data type represents a blob column with a maximum length of 16,777,215 (2^{24} - 1) bytes. In storage, each uses a three-byte length prefix indicating the byte quantity in the value.

- **LONGBLOB** – This data type represents a blob column with a maximum length of 4,294,967,295(2^{32} - 1) bytes. In storage, each uses a four-byte length prefix indicating the byte quantity in the value.

- **TINYTEXT** – This data type represents a text column with a maximum length of 255 (2^8 - 1) characters. In storage, each uses a one-byte length prefix indicating the byte quantity in the value.

- **TEXT** – This data type represents a text column with a maximum length of 65,535 (2^{16} - 1) characters. In storage, each uses a two-byte length prefix indicating the byte quantity in the value.

- **MEDIUMTEXT** – This data type represents a text column with a maximum length of 16,777,215 (2^{24} - 1) characters. In storage, each uses a three-byte length prefix indicating the byte quantity in the value.

- **LONGTEXT** – This data type represents a text column with a maximum length of 4,294,967,295 or 4GB (2^{32} - 1) characters. In storage, each uses a four-byte length prefix indicating the byte quantity in the value.

- **ENUM** – This data type represents a string object having only a single value from a list.

- **SET** – This data type represents a string object having zero or more values from a list, with a maximum of 64 members. SET values present internally as integer values.

10. MariaDB – Create Tables

In this chapter, we will learn how to create tables. Before creating a table, first determine its name, field names, and field definitions.

Following is the general syntax for table creation-

```
CREATE TABLE table_name (column_name column_type);
```

Review the command applied to creating a table in the PRODUCTS database-

```
products_tbl(
   product_id INT NOT NULL AUTO_INCREMENT,
   product_name VARCHAR(100) NOT NULL,
   product_manufacturer VARCHAR(40) NOT NULL,
   submission_date DATE,
   PRIMARY KEY ( product_id )
);
```

The above example uses "NOT NULL" as a field attribute to avoid errors caused by a null value. The attribute "AUTO_INCREMENT" instructs MariaDB to add the next available value to the ID field. The keyword **primary key** defines a column as the primary key. Multiple columns separated by commas can define a primary key.

The two main methods for creating tables are using the command prompt and a PHP script.

The Command Prompt

Utilize the CREATE TABLE command to perform the task as shown below-

```
root@host# mysql -u root -p
Enter password:*******
mysql> use PRODUCTS;
Database changed
mysql> CREATE TABLE products_tbl(
   -> product_id INT NOT NULL AUTO_INCREMENT,
   -> product_name VARCHAR(100) NOT NULL,
   -> product_manufacturer VARCHAR(40) NOT NULL,
   -> submission_date DATE,
   -> PRIMARY KEY ( product_id )
   -> );
```

```
mysql> SHOW TABLES;
```

```
+--------------------+
| PRODUCTS           |
+--------------------+
| products_tbl       |
+--------------------+
```

Ensure all commands are terminated with a semicolon.

PHP Create Table Script

PHP provides **mysql_query()** for table creation. Its second argument contains the necessary SQL command-

```
<html>
<head>
<title>Create a MariaDB Table</title>
</head>
<body>
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully<br />';
$sql = "CREATE TABLE products_tbl( ".
       "product_id INT NOT NULL AUTO_INCREMENT, ".
       "product_name VARCHAR(100) NOT NULL, ".
       "product_manufacturer VARCHAR(40) NOT NULL, ".
       "submission_date DATE, ".
       "PRIMARY KEY ( product_id )); ";
mysql_select_db( 'PRODUCTS' );
$retval = mysql_query( $sql, $conn );
if(! $retval )
```

```
{
  die('Could not create table: ' . mysql_error());
}
echo "Table created successfully\n";
mysql_close($conn);
?>
</body>
</html>
```

On successful table creation, you will see the following output:

```
mysql> Table created successfully
```

11. MariaDB – Drop Tables

In this chapter, we will learn to delete tables.

Table deletion is very easy, but remember all deleted tables are irrecoverable. The general syntax for table deletion is as follows-

```
DROP TABLE table_name ;
```

Two options exist for performing a table drop: use the command prompt or a PHP script.

The Command Prompt

At the command prompt, simply use the **DROP TABLE** SQL command-

```
root@host# mysql -u root -p
Enter password:*******
mysql> use PRODUCTS;
Database changed
mysql> DROP TABLE products_tbl

mysql> SELECT * from products_tbl
ERROR 1146 (42S02): Table 'products_tbl' doesn't exist
```

PHP Drop Table Script

PHP provides **mysql_query()** for dropping tables. Simply pass its second argument the appropriate SQL command-

```
<html>
<head>
<title>Create a MariaDB Table</title>
</head>
<body>
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
```

```
{
  die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully<br />';
$sql = "DROP TABLE products_tbl";
mysql_select_db( 'PRODUCTS' );
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not delete table: ' . mysql_error());
}
echo "Table deleted successfully\n";
mysql_close($conn);
?>
</body>
</html>
```

On successful table deletion, you will see the following output:

```
mysql> Table deleted successfully
```

12. MariaDB – Insert Query

In this chapter, we will learn how to insert data in a table.

Inserting data into a table requires the INSERT command. The general syntax of the command is INSERT followed by the table name, fields, and values.

Review its general syntax given below-

```
INSERT INTO tablename (field,field2,...) VALUES (value, value2,...);
```

The statement requires the use of single or double quotes for string values. Other options for the statement include "INSERT...SET" statements, "INSERT...SELECT" statements, and several other options.

Note: The VALUES() function that appears within the statement, only applies to INSERT statements and returns NULL if used elsewhere.

Two options exist for performing the operation: use the command line or use a PHP script.

The Command Prompt

At the prompt, there are many ways to perform a select operation. A standard statement is given below-

```
mysql> INSERT INTO products_tbl (ID_number, Nomenclature) VALUES (12345,
"Orbitron 4000");


mysql> SHOW COLUMNS FROM products_tbl;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| ID_number   | int(5)      |      |     |         |       |
| Nomenclature| char(13)    |      |     |         |       |
+-------------+-------------+------+-----+---------+-------+
```

You can insert multiple rows-

```
INSERT INTO products VALUES (1, "first row"), (2, "second row");
```

You can also employ the SET clause-

```
INSERT INTO products SELECT * FROM inventory WHERE status = 'available';
```

PHP Insertion Script

Employ the same "INSERT INTO..." statement within a PHP function to perform the operation. You will use the **mysql_query()** function once again.

Review the example given below-

```
<?php
if(isset($_POST['add']))
{
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}

if(! get_magic_quotes_gpc() )
{
   $product_name = addslashes ($_POST['product_name']);
   $product_manufacturer = addslashes ($_POST['product_name']);
}
else
{
   $product_name = $_POST['product_name'];
   $product_manufacturer = $_POST['product_manufacturer'];
}
$ship_date = $_POST['ship_date'];

$sql = "INSERT INTO products_tbl ".
       "(product_name,product_manufacturer, ship_date) ".
       "VALUES ".
       "('$product_name','$product_manufacturer','$ship_date')";
mysql_select_db('PRODUCTS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
```

```
  die('Could not enter data: ' . mysql_error());
}
echo "Entered data successfully\n";
mysql_close($conn);
}
else
{
?>
```

On successful data insertion, you will see the following output:

```
mysql> Entered data successfully
```

You will also collaborate validation statements with insert statements such as checking to ensure correct data entry. MariaDB includes a number of options for this purpose, some of which are automatic.

13. MariaDB – Select Query

In this chapter, we will learn how to select data from a table.

SELECT statements retrieve selected rows. They can include UNION statements, an ordering clause, a LIMIT clause, a WHERE clause, a GROUP BY...HAVING clause, and subqueries.

Review the following general syntax-

```
SELECT field, field2,... FROM table_name, table_name2,... WHERE...
```

A SELECT statement provides multiple options for specifying the table used-

- database_name.table_name
- table_name.column_name
- database_name.table_name.column_name

All select statements must contain one or more **select expressions**. Select expressions consist of one of the following options-

- A column name.
- An expression employing operators and functions.
- The specification "table_name.*" to select all columns within the given table.
- The character "*" to select all columns from all tables specified in the FROM clause.

The command prompt or a PHP script can be employed in executing a select statement.

The Command Prompt

At the command prompt, execute statements as follows-

```
root@host# mysql -u root -p password;
Enter password:*******

mysql> use PRODUCTS;
Database changed

mysql> SELECT * from products_tbl

+-------------+---------------+
| ID_number   | Nomenclature  |
+-------------+---------------+
```

```
| 12345       | Orbitron 4000 |
+-------------+---------------+
```

PHP Select Script

Employ the same SELECT statement(s) within a PHP function to perform the operation. You will use the **mysql_query()** function once again. Review an example given below-

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
$sql = 'SELECT product_id, product_name,
                product_manufacturer, ship_date
        FROM products_tbl';

mysql_select_db('PRODUCTS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not get data: ' . mysql_error());
}
while($row = mysql_fetch_array($retval, MYSQL_ASSOC))
{
    echo "Product ID :{$row['product_id']}  <br> ".
         "Name: {$row['product_name']} <br> ".
         "Manufacturer: {$row['product_manufacturer']} <br> ".
         "Ship Date : {$row['ship_date']} <br> ".
         "--------------------------------<br>";
}
echo "Fetched data successfully\n";
mysql_close($conn);
```

```
?>
```

On successful data retrieval, you will see the following output:

```
Product ID: 12345
Nomenclature: Orbitron 4000
Manufacturer: XYZ Corp
Ship Date: 01/01/17
-------------------------------------------
Product ID: 12346
Nomenclature: Orbitron 3000
Manufacturer: XYZ Corp
Ship Date: 01/02/17
-------------------------------------------
mysql> Fetched data successfully
```

Best practices suggest releasing cursor memory after every SELECT statement. PHP provides the **mysql_free_result()** function for this purpose. Review its use as shown below-

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
$sql = 'SELECT product_id, product_name,
                product_manufacturer, ship_date
        FROM products_tbl';

mysql_select_db('PRODUCTS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not get data: ' . mysql_error());
}
```

```
while($row = mysql_fetch_array($retval, MYSQL_NUM))
{
    echo "Product ID :{$row[0]}  <br> ".
         "Name: {$row[1]} <br> ".
         "Manufacturer: {$row[2]} <br> ".
         "Ship Date : {$row[3]} <br> ".
         "--------------------------------<br>";
}
mysql_free_result($retval);
echo "Fetched data successfully\n";
mysql_close($conn);
?>
```

14. MariaDB – Where Clause

WHERE clauses filter various statements such as SELECT, UPDATE, DELETE, and INSERT. They present criteria used to specify action. They typically appear after a table name in a statement, and their condition follows. The WHERE clause essentially functions like an **if** statement.

Review the general syntax of WHERE clause given below-

```
[COMMAND] field,field2,... FROM table_name,table_name2,... WHERE [CONDITION]
```

Note the following qualities of the WHERE clause-

- It is optional.
- It allows any condition to be specified.
- It allows for the specification of multiple conditions through using an AND or OR operator.
- Case sensitivity only applies to statements using LIKE comparisons.

The WHERE clause permits the use of the following operators-

Operator
= !=
> <
>= <=

WHERE clauses can be utilized at the command prompt or within a PHP script.

The Command Prompt

At the command prompt, simply use a standard command-

```
root@host# mysql -u root -p password;
Enter password:*******
mysql> use PRODUCTS;
Database changed
mysql> SELECT * from products_tbl WHERE product_manufacturer='XYZ Corp';
+-------------+----------------+----------------------+
| ID_number   | Nomenclature   | product_manufacturer |
+-------------+----------------+----------------------+
|      12345  | Orbitron 4000  | XYZ Corp             |
```

```
+-------------+---------------+---------------------+
|      12346  | Orbitron 3000  | XYZ Corp            |
+-------------+---------------+---------------------+
|      12347  | Orbitron 1000  | XYZ Corp            |
+-------------+---------------+---------------------+
```

Review an example using the **AND** condition-

```
SELECT *
FROM products_tbl
WHERE product_name = 'Bun Janshu 3000';
AND product_id <= 344;
```

This example combines both AND and OR conditions:

```
SELECT *
FROM products_tbl
WHERE (product_name = 'Bun Janshu 3000' AND product_id < 344)
OR (product_name = 'Bun Janshu 3000');
```

PHP Scripts Using Where Clause

Employ the **mysql_query()** function in operations using a WHERE clause-

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
$sql = 'SELECT product_id, product_name,
                product_manufacturer, ship_date
        FROM products_tbl
        WHERE product_manufacturer="XYZ Corp"';

mysql_select_db('PRODUCTS');
```

```
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not get data: ' . mysql_error());
}
while($row = mysql_fetch_array($retval, MYSQL_ASSOC))
{
    echo "Product ID :{$row['product_id']}  <br> ".
         "Name: {$row['product_name']} <br> ".
         "Manufacturer: {$row['product_manufacturer']} <br> ".
         "Ship Date: {$row['ship_date']} <br> ".
         "--------------------------------<br>";
}
echo "Fetched data successfully\n";
mysql_close($conn);
?>
```

On successful data retrieval, you will see the following output:

```
Product ID: 12345
Nomenclature: Orbitron 4000
Manufacturer: XYZ Corp
Ship Date: 01/01/17
--------------------------------------------
Product ID: 12346
Nomenclature: Orbitron 3000
Manufacturer: XYZ Corp
Ship Date: 01/02/17
--------------------------------------------
Product ID: 12347
Nomenclature: Orbitron 1000
Manufacturer: XYZ Corp
Ship Date: 01/02/17
--------------------------------------------
mysql> Fetched data successfully
```

15. MariaDB – Update Query

The **UPDATE** command modifies existing fields by changing values. It uses the SET clause to specify columns for modification, and to specify the new values assigned. These values can be either an expression or the default value of the field. Setting a default value requires using the DEFAULT keyword. The command can also employ a WHERE clause to specify conditions for an update and/or an ORDER BY clause to update in a certain order.

Review the following general syntax-

```
UPDATE table_name SET field=new_value, field2=new_value2,...
[WHERE ...]
```

Execute an UPDATE command from either the command prompt or using a PHP script.

The Command Prompt

At the command prompt, simply use a standard command-

```
root@host# mysql -u root -p password;
Enter password:*******
mysql> use PRODUCTS;
Database changed
mysql> UPDATE products_tbl
       SET nomenclature='Fiber Blaster 300Z'
       WHERE ID_number=112;

mysql> SELECT * from products_tbl WHERE ID_number='112';
+-------------+--------------------+----------------------+
| ID_number   | Nomenclature       | product_manufacturer |
+-------------+--------------------+----------------------+
|      112    | Fiber Blaster 300Z | XYZ Corp             |
+-------------+--------------------+----------------------+
```

PHP Update Query Script

Employ the **mysql_query()** function in UPDATE command statements-

```
<?php
$dbhost = 'localhost:3036';
```

```
$dbuser = 'root';
$dbpass = 'rootpassword';

$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
$sql = 'UPDATE products_tbl
        SET product_name="Fiber Blaster 300z"
        WHERE product_id=112';

mysql_select_db('PRODUCTS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not update data: ' . mysql_error());
}
echo "Updated data successfully\n";
mysql_close($conn);
?>
```

On successful data update, you will see the following output:

```
mysql> Updated data successfully
```

16. MariaDB – Delete Query

The DELETE command deletes table rows from the specified table, and returns the quantity deleted. Access the quantity deleted with the ROW_COUNT() function. A WHERE clause specifies rows, and in its absence, all rows are deleted. A LIMIT clause controls the number of rows deleted.

In a DELETE statement for multiple rows, it deletes only those rows satisfying a condition; and LIMIT and WHERE clauses are not permitted. DELETE statements allow deleting rows from tables in different databases, but do not allow deleting from a table and then selecting from the same table within a subquery.

Review the following DELETE syntax-

```
DELETE FROM table_name [WHERE …]
```

Execute a DELETE command from either the command prompt or using a PHP script.

The Command Prompt

At the command prompt, simply use a standard command-

```
root@host# mysql -u root -p password;
Enter password:*******
mysql> use PRODUCTS;
Database changed
mysql> DELETE FROM products_tbl WHERE product_id=133;
mysql> SELECT * from products_tbl WHERE ID_number='133';
ERROR 1032 (HY000): Can't find record in 'products_tbl'
```

PHP Delete Query Script

Use the **mysql_query()** function in DELETE command statements-

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
```

```
}
$sql = 'DELETE FROM products_tbl
        WHERE product_id=261';

mysql_select_db('PRODUCTS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not delete data: ' . mysql_error());
}
echo "Deleted data successfully\n";
mysql_close($conn);
?>
```

On successful data deletion, you will see the following output:

```
mysql> Deleted data successfully
mysql> SELECT * from products_tbl WHERE ID_number='261';
ERROR 1032 (HY000): Can't find record in 'products_tbl'
```

17. MariaDB – Like Clause

The WHERE clause provides a way to retrieve data when an operation uses an exact match. In situations requiring multiple results with shared characteristics, the **LIKE** clause accommodates broad pattern matching.

A LIKE clause tests for a pattern match, returning a true or false. The patterns used for comparison accept the following wildcard characters: "%", which matches numbers of characters (0 or more); and "_", which matches a single character. The "_" wildcard character only matches characters within its set, meaning it will ignore latin characters when using another set. The matches are case-insensitive by default requiring additional settings for case sensitivity.

A NOT LIKE clause allows for testing the opposite condition, much like the **not** operator.

If the statement expression or pattern evaluate to NULL, the result is NULL.

Review the general LIKE clause syntax given below-

```
SELECT field, field2,... FROM table_name, table_name2,...
WHERE field LIKE condition
```

Employ a LIKE clause either at the command prompt or within a PHP script.

The Command Prompt

At the command prompt, simply use a standard command-

```
root@host# mysql -u root -p password;
Enter password:*******
mysql> use TUTORIALS;
Database changed

mysql> SELECT * from products_tbl
       WHERE product_manufacturer LIKE 'XYZ%';
+-------------+----------------+----------------------+
| ID_number   | Nomenclature   | product_manufacturer |
+-------------+----------------+----------------------+
|      12345  | Orbitron 4000  | XYZ Corp             |
+-------------+----------------+----------------------+
|      12346  | Orbitron 3000  | XYZ Corp             |
+-------------+----------------+----------------------+
|      12347  | Orbitron 1000  | XYZ Corp             |
```

```
+-------------+----------------+---------------------+
```

PHP Script Using Like Clause

Use the **mysql_query()** function in statements employing the LIKE clause-

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
$sql = 'SELECT product_id, product_name,
               product_manufacturer, ship_date
        FROM products_tbl
        WHERE product_manufacturer LIKE "xyz%"';

mysql_select_db('PRODUCTS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not get data: ' . mysql_error());
}
while($row = mysql_fetch_array($retval, MYSQL_ASSOC))
{
    echo "Product ID:{$row['product_id']}  <br> ".
         "Name: {$row['product_name']} <br> ".
         "Manufacturer: {$row['product_manufacturer']} <br> ".
         "Ship Date: {$row['ship_date']} <br> ".
         "--------------------------------<br>";
}
echo "Fetched data successfully\n";
mysql_close($conn);
?>
```

On successful data retrieval, you will see the following output:

```
Product ID: 12345
Nomenclature: Orbitron 4000
Manufacturer: XYZ Corp
Ship Date: 01/01/17
-----------------------------------------------
Product ID: 12346
Nomenclature: Orbitron 3000
Manufacturer: XYZ Corp
Ship Date: 01/02/17
-----------------------------------------------
Product ID: 12347
Nomenclature: Orbitron 1000
Manufacturer: XYZ Corp
Ship Date: 01/02/17
-----------------------------------------------
mysql> Fetched data successfully
```

18. MariaDB – Order By Clause

The **ORDER BY** clause, as mentioned in previous discussions, sorts the results of a statement. It specifies the order of the data operated on, and includes the option to sort in ascending (ASC) or descending (DESC) order. On omission of order specification, the default order is ascending.

ORDER BY clauses appear in a wide variety of statements such as DELETE and UPDATE. They always appear at the end of a statement, not in a subquery or before a set function, because they operate on the final resulting table. You also cannot use an integer to identify a column.

Review the general syntax of the ORDER BY clause given below-

```
SELECT field, field2,... [or column] FROM table_name, table_name2,...
ORDER BY field, field2,... ASC[or DESC]
```

Use an ORDER BY clause either at the command prompt or within a PHP script.

The Command Prompt

At the command prompt, simply use a standard command-

```
root@host# mysql -u root -p password;
Enter password:*******
mysql> use PRODUCTS;
Database changed

mysql> SELECT * from products_tbl ORDER BY product_manufacturer ASC
+-------------+----------------+----------------------+
| ID_number   | Nomenclature   | product_manufacturer |
+-------------+----------------+----------------------+
|      56789  | SuperBlast 400 | LMN Corp             |
+-------------+----------------+----------------------+
|      67891  | Zoomzoom 5000  | QFT Corp             |
+-------------+----------------+----------------------+
|      12347  | Orbitron 1000  | XYZ Corp             |
+-------------+----------------+----------------------+
```

PHP Script Using Order By Clause

Utilize the **mysql_query()** function, once again, in statements employing the ORDER BY clause-

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
$sql = 'SELECT product_id, product_name,
               product_manufacturer, ship_date
        FROM products_tbl
        ORDER BY  product_manufacturer DESC';

mysql_select_db('PRODUCTS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not get data: ' . mysql_error());
}
while($row = mysql_fetch_array($retval, MYSQL_ASSOC))
{
    echo "Product ID :{$row['product_id']}  <br> ".
         "Name: {$row['product_name']} <br> ".
         "Manufacturer: {$row['product_manufacturer']} <br> ".
         "Ship Date : {$row['ship_date']} <br> ".
         "--------------------------------<br>";
}
echo "Fetched data successfully\n";
mysql_close($conn);
?>
```

On successful data retrieval, you will see the following output:

```
Product ID: 12347
Nomenclature: Orbitron 1000
Manufacturer: XYZ Corp
Ship Date: 01/01/17
-----------------------------------------------
Product ID:  67891
Nomenclature:  Zoomzoom 5000
Manufacturer:  QFT Corp
Ship Date: 01/01/17
-----------------------------------------------
Product ID:  56789
Nomenclature:  SuperBlast 400
Manufacturer:  LMN Corp
Ship Date: 01/04/17
-----------------------------------------------
mysql> Fetched data successfully
```

19. MariaDB – Join

In previous discussions and examples, we examined retrieving from a single table, or retrieving multiple values from multiple sources. Most real-world data operations are much more complex, requiring aggregation, comparison, and retrieval from multiple tables.

JOINs allow merging of two or more tables into a single object. They are employed through SELECT, UPDATE, and DELETE statements.

Review the general syntax of a statement employing a JOIN as shown below-

```
SELECT column
FROM table_name1
INNER JOIN table_name2
ON table_name1.column = table_name2.column;
```

Note the old syntax for JOINS used implicit joins and no keywords. It is possible to use a WHERE clause to achieve a join, but keywords work best for readability, maintenance, and best practices.

JOINs come in many forms such as a left join, right join, or inner join. Various join types offer different types of aggregation based on shared values or characteristics.

Employ a JOIN either at the command prompt or with a PHP script.

The Command Prompt

At the command prompt, simply use a standard statement-

```
root@host# mysql -u root -p password;
Enter password:*******
mysql> use PRODUCTS;
Database changed

mysql> SELECT products.ID_number, products.Nomenclature, inventory.inventory_ct
       FROM products
       INNER JOIN inventory
       ON products.ID_number = inventory.ID_number;
+-------------+----------------+-----------------+
| ID_number   | Nomenclature   | Inventory Count |
+-------------+----------------+-----------------+
|      12345  | Orbitron 4000  | 150             |
+-------------+----------------+-----------------+
```

```
|      12346  | Orbitron 3000  | 200             |
+-------------+----------------+-----------------+
|      12347  | Orbitron 1000  | 0               |
+-------------+----------------+-----------------+
```

PHP Script Using JOIN

Use the **mysql_query()** function to perform a join operation-

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
$sql = 'SELECT a.product_id, a.product_manufacturer, b.product_count
        FROM products_tbl a, pcount_tbl b
        WHERE a.product_manufacturer = b.product_manufacturer';

mysql_select_db('PRODUCTS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not get data: ' . mysql_error());
}
while($row = mysql_fetch_array($retval, MYSQL_ASSOC))
{
    echo "Manufacturer:{$row['product_manufacturer']}  <br> ".
         "Count: {$row['product_count']} <br> ".
         "Product ID: {$row['product_id']} <br> ".
         "--------------------------------<br>";
}
echo "Fetched data successfully\n";
mysql_close($conn);
```

```
?>
```

On successful data retrieval, you will see the following output:

```
ID Number: 12345
Nomenclature: Orbitron 4000
Inventory Count: 150
---------------------------------------
ID Number: 12346
Nomenclature: Orbitron 3000
Inventory Count: 200
---------------------------------------
ID Number: 12347
Nomenclature: Orbitron 1000
Inventory Count: 0
---------------------------------------
mysql> Fetched data successfully
```

20. MariaDB – Null Values

When working with NULL values, remember they are unknown values. They are not empty strings or zero, which are valid values. In table creation, column specifications allow for setting them to accept null values, or reject them. Simply utilize a NULL or NOT NULL clause. This has applications in cases of missing record information like an ID number.

User-defined variables have a value of NULL until explicit assignment. Stored routine parameters and local variables allow setting a value of NULL. When a local variable has no default value, it has a value of NULL.

NULL is case-insensitive, and has the following aliases-

- `UNKNOWN` (a boolean value)
- `\N`

NULL Operators

Standard comparison operators cannot be used with NULL (e.g., =, >, >=, <=, <, or !=) because all comparisons with a NULL value return NULL, not true or false. Comparisons with NULL or possibly containing it must use the "`<=>`" (NULL-SAFE) operator.

Other available operators are-

- **`IS NULL`** – It tests for a NULL value.
- **`IS NOT NULL`** – It confirms the absence of a NULL value.
- **`ISNULL`** – It returns a value of 1 on discovery of a NULL value, and 0 in its absence.
- **`COALESCE`** – It returns the first non-NULL value of a list, or it returns a NULL value in the absence of one.

Sorting NULL Values

In sorting operations, NULL values have the lowest value, so DESC order results in NULL values at the bottom. MariaDB allows for setting a higher value for NULL values.

There are two ways to do this as shown below-

```
SELECT column1 FROM product_tbl ORDER BY ISNULL(column1), column1;
```

The other way-

```
SELECT column1 FROM product_tbl ORDER BY IF(column1 IS NULL, 0, 1), column1 DESC;
```

NULL Functions

Functions generally output NULL when any parameters are NULL. However, there are functions specifically designed for managing NULL values. They are-

- **IFNULL() –** If the first expression is not NULL it returns it. When it evaluates to NULL, it returns the second expression.

- **NULLIF() –** It returns NULL when the compared expressions are equal, if not, it returns the first expression.

Functions like SUM and AVG ignore NULL values.

Inserting NULL Values

On insertion of a NULL value in a column declared NOT NULL, an error occurs. In default SQL mode, a NOT NULL column will instead insert a default value based on data type.

When a field is a TIMESTAMP, AUTO_INCREMENT, or virtual column, MariaDB manages NULL values differently. Insertion in an AUTO_INCREMENT column causes the next number in the sequence to insert in its place. In a TIMESTAMP field, MariaDB assigns the current timestamp instead. In virtual columns, a topic discussed later in this tutorial, the default value is assigned.

UNIQUE indices can hold many NULL values, however, primary keys cannot be NULL.

NULL Values and the Alter Command

When you use the ALTER command to modify a column, in the absence of NULL specifications, MariaDB automatically assigns values.

21. MariaDB – Regular Expression

Beyond the pattern matching available from LIKE clauses, MariaDB offers regular expression-based matching through the REGEXP operator. The operator performs pattern matching for a string expression based on a given pattern.

MariaDB 10.0.5 introduced PCRE Regular Expressions, which dramatically increases the scope of matching into areas like recursive patterns, look-ahead assertions, and more.

Review the use of standard REGEXP operator syntax given below-

```
SELECT column FROM table_name WHERE column REGEXP '[PATTERN]';
```

REGEXP returns 1 for a pattern match or 0 in the absence of one.

An option for the opposite exists in the form of NOT REGEXP. MariaDB also offers synonyms for REGEXP and NOT REGEXP, RLIKE and NOT RLIKE, which were created for compatibility reasons.

The pattern compared can be a literal string or something else such as a table column. In strings, it uses C escape syntax, so double any "\" characters. REGEXP is also case-insensitive, with the exception of binary strings.

A table of possible patterns, which can be used are given below-

Pattern	Description
^	It matches the start of the string.
$	It matches the string's end.
.	It matches a single character.
[...]	It matches any character in the brackets.
[^...]	It matches any character not listed in the brackets.
p1|p2|p3	It matches any of the patterns.
*	It matches 0 or more instances of the preceding element.
+	It matches 1 or more instances of the preceding element.
{n}	It matches n instances of the preceding element.
{m,n}	It matches m to n instances of the preceding element.

Review the pattern matching examples given below-

Products starting with "pr"-

```
SELECT name FROM product_tbl WHERE name REGEXP '^pr';
```

Products ending with "na"-

```
SELECT name FROM product_tbl WHERE name REGEXP 'na$';
```

Products starting with a vowel-

```
SELECT name FROM product_tbl WHERE name REGEXP '^[aeiou]';
```

22. MariaDB – Transactions

Transactions are sequential group operations. They function as a single unit, and do not terminate until all operations within the group execute successfully. A single failure in the group causes the entire transaction to fail, and causes it to have no impact on the database.

Transactions conform to ACID (Atomicity, Consistency, Isolation, and Durability):

- **Atomicity** – It ensures the success of all operations by aborting on failures and rolling back changes.
- **Consistency** – It ensures the database applies changes on a successful transaction.
- **Isolation** – It enables independent transactions operation of transactions.
- **Durability** – It ensures the persistence of a successful transaction in the event of system failure.

At the head of a transaction statement is the START TRANSACTION statement followed by COMMIT and ROLLBACK statements-

- START TRANSACTION begins the transaction.
- COMMIT saves changes to data.
- ROLLBACK ends the transaction, destroying any changes.

On a successful transaction, COMMIT acts. On a failure, ROLLBACK acts.

Note: Some statements cause an implicit commit, and they also cause an error when used within transactions. Examples of such statements include, but are not limited to CREATE, ALTER, and DROP.

MariaDB transactions also include options like SAVEPOINT and LOCK TABLES. SAVEPOINT sets a restore point to utilize with ROLLBACK. LOCK TABLES allows controlling access to tables during sessions to prevent modifications during certain time periods.

The AUTOCOMMIT variable provides control over transactions. A setting of 1 forces all operations to be considered successful transactions, and a setting of 0 causes persistence of changes to only occur on an explicit COMMIT statement.

Structure of a Transaction

The general structure of a transaction statement consists of beginning with START TRANSACTION. The next step is inserting one or more commands/operations, inserting statements that check for errors, inserting ROLLBACK statements to manage any errors discovered and finally inserting a COMMIT statement to apply changes on successful operations.

Review the example given below-

```
START TRANSACTION;
SELECT name FROM products WHERE manufacturer='XYZ Corp';
UPDATE spring_products SET item=name;
COMMIT;
```

23. MariaDB – Alter Command

The ALTER command provides a way to change an existing table's structure, meaning modifications like removing or adding columns, modifying indices, changing data types, or changing names. ALTER also waits to apply changes when a metadata lock is active.

Using ALTER to Modify Columns

ALTER paired with DROP removes an existing column. However, it fails if the column is the only remaining column.

Review the example given below-

```
mysql> ALTER TABLE products_tbl  DROP version_num;
```

Use an ALTER...ADD statement to add columns-

```
mysql> ALTER TABLE products_tbl ADD discontinued CHAR(1);
```

Use the keywords FIRST and AFTER to specify placement of the column-

```
ALTER TABLE products_tbl ADD discontinued CHAR(1) FIRST;

ALTER TABLE products_tbl ADD discontinued CHAR(1) AFTER quantity;
```

Note the FIRST and AFTER keywords only apply to ALTER...ADD statements. Furthermore, you must drop a table and then add it in order to reposition it.

Change a column definition or name by using the MODIFY or CHANGE clause in an ALTER statement. The clauses have similar effects, but utilize substantially different syntax.

Review a CHANGE example given below-

```
mysql> ALTER TABLE products_tbl CHANGE discontinued status CHAR(4);
```

In a statement using CHANGE, specify the original column and then the new column that will replace it. Review a MODIFY example below:

```
mysql> ALTER TABLE products_tbl MODIFY discontinued CHAR(4);
```

The ALTER command also allows for changing default values. Review an example:

```
mysql> ALTER TABLE products_tbl ALTER discontinued SET DEFAULT N;
```

You can also use it to remove default constraints by pairing it with a DROP clause:

```
mysql> ALTER TABLE products_tbl ALTER discontinued DROP DEFAULT;
```

Using ALTER to Modify Tables

Change table type with the TYPE clause-

```
mysql> ALTER TABLE products_tbl TYPE = INNODB;
```

Rename a table with the RENAME keyword-

```
mysql> ALTER TABLE products_tbl RENAME TO products2016_tbl;
```

24. MariaDB – Indexes and Statistics Tables

Indexes are tools for accelerating record retrieval. An index spawns an entry for each value within an indexed column.

There are four types of indexes-

- **Primary** (one record represents all records)
- **Unique** (one record represents multiple records)
- **Plain**
- **Full-Text** (permits many options in text searches).

The terms "key" and "index" are identical in this usage.

Indexes associate with one or more columns, and support rapid searches and efficient record organization. When creating an index, consider which columns are frequently used in your queries. Then create one or multiple indexes on them. In addition, view indexes as essentially tables of primary keys.

Though indexes accelerate searches or SELECT statements, they make insertions and updates drag due to performing the operations on both the tables and the indexes.

Create an Index

You can create an index through a CREATE TABLE...INDEX statement or a CREATE INDEX statement. The best option supporting readability, maintenance, and best practices is CREATE INDEX.

Review the general syntax of Index given below-

```
CREATE [UNIQUE or FULLTEXT or...] INDEX index_name ON table_name column;
```

Review an example of its use-

```
CREATE UNIQUE INDEX top_sellers ON products_tbl product;
```

Drop an Index

You can drop an index with DROP INDEX or ALTER TABLE...DROP. The best option supporting readability, maintenance, and best practices is DROP INDEX.

Review the general syntax of Drop Index given below-

```
DROP INDEX index_name ON table_name;
```

Review an example of its use-

```
DROP INDEX top_sellers ON product_tbl;
```

Rename an Index

Rename an index with the ALTER TABLE statement. Review its general syntax given below-

```
ALTER TABLE table_name DROP INDEX index_name, ADD INDEX new_index_name;
```

Review an example of its use-

```
ALTER TABLE products_tbl DROP INDEX top_sellers, ADD INDEX top_2016sellers;
```

Managing Indexes

You will need to examine and track all indexes. Use SHOW INDEX to list all existing indexes associated with a given table. You can set the format of the displayed content by using an option such as "\G", which specifies a vertical format.

Review the following example-

```
mysql> SHOW INDEX FROM products_tbl\G
```

Table Statistics

Indexes are used heavily to optimize queries given the faster access to records, and the statistics provided. However, many users find index maintenance cumbersome. MariaDB 10.0 made storage engine independent statistics tables available, which calculate data statistics for every table in every storage engine, and even statistics for columns that are not indexed.

25. MariaDB – Temporary Tables

Some operations can benefit from temporary tables due to speed or disposable data. The life of a temporary table ends at the termination of a session whether you employ them from the command prompt, with a PHP script, or through a client program. It also does not appear in the system in a typical fashion. The SHOW TABLES command will not reveal a list containing temporary tables.

Create a Temporary Table

The TEMPORARY keyword within a CREATE TABLE statement spawns a temporary table. Review an example given below-

```
mysql> CREATE TEMPORARY TABLE order (
     item_name VARCHAR(50) NOT NULL
     , price DECIMAL(7,2) NOT NULL DEFAULT 0.00
     , quantity INT UNSIGNED NOT NULL DEFAULT 0
);
```

In creating a temporary table, you can clone existing tables, meaning all their general characteristics, with the LIKE clause. The CREATE TABLE statement used to spawn the temporary table will not commit transactions as a result of the TEMPORARY keyword.

Though temporary tables stand apart from non-temporary and drop at the end of a session, they may have certain conflicts-

- They sometimes conflict with ghost temporary tables from expired sessions.
- They sometimes conflict with shadow names of non-temporary tables.

Note: Temporary tables are permitted to have the same name as an existing non-temporary table because MariaDB views it as a difference reference.

Administration

MariaDB requires granting privileges to users for creating temporary tables. Utilize a GRANT statement to give this privilege to non-admin users.

```
GRANT CREATE TEMPORARY TABLES ON orders TO 'machine122'@'localhost';
```

Drop a Temporary Table

Though temporary tables are essentially removed at the end of sessions, you have the option to delete them. Dropping a temporary table requires the use of the TEMPORARY keyword, and best practices suggest dropping temporary tables before any non-temporary.

```
mysql> DROP TABLE order;
```

26. MariaDB – Table Cloning

Some situations require producing an exact copy of an existing table. The CREATE...SELECT statement cannot produce this output because it neglects things like indexes and default values.

The procedure for a duplicating a table is as follows-

- Utilize SHOW CREATE TABLE to produce a CREATE TABLE statement that details the entire structure of the source table.
- Edit the statement to give the table a new name, and execute it.
- Use an INSERT INTO...SELECT statement if you also need the table data copied.

```
mysql> INSERT INTO inventory_copy_tbl (product_id,
                              product_name,
                              product_manufacturer,
                              ship_date)
     SELECT product_id,product_name,
            product_manufacturer,ship_date,
     FROM inventory_tbl;
```

Another method for creating a duplicate uses a CREATE TABLE AS statement. The statement copies all columns, column definitions, and populates the copy with the source table's data.

Review its syntax given below-

```
CREATE TABLE clone_tbl AS
  SELECT columns
  FROM original_tbl
  [WHERE conditions];
```

Review an example of its use below-

```
CREATE TABLE products_copy_tbl AS
  SELECT *
  FROM products_tbl;
```

27. MariaDB – Sequences

In version 10.0.3, MariaDB introduced a storage engine known as sequence. Its ad hoc generates an integer sequence for operations, and then it terminates. The sequence contains positive integers in descending or ascending order, and uses a starting, ending, and increment value.

It does not allow use in multiple queries, only in its original query because of its virtual (not written to disk) nature. However, sequence tables can be converted to standard tables through an ALTER command. If a converted table is deleted, the sequence table still exists. Sequences also cannot produce negative numbers or rotate at the minimum/maximum.

Installating the Sequence Engine

Using sequences requires installing the sequence engine, which MariaDB distributes as a plugin rather than binary. Install it with the following command-

```
INSTALL SONAME "ha_sequence";
```

After installation, verify it-

```
SHOW ENGINES\G
```

Remember that after engine installation, you cannot create a standard table with a name that uses sequence syntax, but you can create a temporary table with a sequence-syntax name.

Creating Sequence

There are two methods of sequence creation-

- Create a table and use the AUTO_INCREMENT attribute to define a column as auto-increment.

- Use an existing database and use a sequence SELECT query to produce a sequence. The query uses seq_ [FROM] _to_[TO] or seq_[FROM]_to_[TO]_step_STEP syntax.

Best practices prefer the use of the second method. Review an example of a sequence creation given below-

```
SELECT * FROM seq_77_to_99;
```

Sequences have many uses-

- Locate missing values within a column to protect against related issues in operations-

```
SELECT myseq.seq FROM seq_22_to_28 myseq LEFT JOIN table1 t ON myseq.seq
= x.y WHERE x.y IS NULL;
```

- Construct a combination of values-

```
SELECT x1.seq, x2.seq FROM seq_5_to_9 x1 JOIN seq_5_to_9 x2 ORDER BY 5, 6;
```

- Find multiples of a number-

```
SELECT seq FROM seq_3_to_100_step_4;
```

- Construct a date sequence for use in applications like booking systems.
- Construct a time sequence.

28. MariaDB – Managing Duplicates

MariaDB, as discussed in earlier lessons, allows duplicate records and tables in some situations. Some of these duplicates are not in fact duplicates due to distinct data or object types, or as a result of unique lifespan or storage of the operation object. These duplicates also typically pose no problems.

In some situations, duplicates do cause problems, and they often appear due to implicit actions or the lenient policy of a MariaDB command. There are ways to control this issue, find duplicates, delete duplicates, and prevent duplicate creation.

Strategies and Tools

There are four key ways to manage duplicates-

- Fish for them with JOIN, and delete them with a temporary table.
- Use INSERT...ON DUPLICATE KEY UPDATE to update on discovery of a duplicate.
- Use DISTINCT to prune the results of a SELECT statement and remove duplicates.
- Use INSERT IGNORE to stop insertion of duplicates.

Using Join with a Temporary Table

Simply perform a semi-join like an inner join, and then remove the duplicates found with a temporary table.

Using INSERT

When INSERT...ON DUPLICATE KEY UPDATE discovers a duplicate unique or primary key, it performs an update. On discovery of multiple unique keys, it updates only the first. Hence, do not use it on tables with multiple unique indices.

Review the following example, which reveals what happens in a table containing indexed values on insertion into a populated field-

```
INSERT INTO add_dupl VALUES (1,'Apple');
ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'
```

Note: If it finds no key, an INSERT...ON DUPLICATE KEY UPDATE statement executes like a normal insert statement.

Using DISTINCT

DISTINCT clauses remove duplicates from results. The general syntax for a DISTINCT clause is as follows-

```
SELECT DISTINCT fields
FROM table
[WHERE conditions];
```

Note: The results of a statement with a DISTINCT clause-

- When using one expression, it returns unique values for it.
- When using multiple expressions, it returns unique combinations.
- It does not ignore NULL values; thus, results also contain NULLs as unique values.

Review the following statement using a DISTINCT clause for a single expression-

```
SELECT DISTINCT product_id
FROM products
WHERE product_name = 'DustBlaster 5000';
```

Review the following example using multiple expressions-

```
SELECT DISTINCT product_name, product_id
FROM products
WHERE product_id < 30
```

Using INSERT IGNORE

An INSERT IGNORE statement instructs MariaDB to cancel insertion on discovery of a duplicate record. Review an example of its use given below-

```
mysql> INSERT IGNORE INTO customer_tbl (LN, FN)
     VALUES( 'Lex', 'Luther');
```

Also, note the logic behind duplicates. Some tables require duplicates based on the nature of that table data. Accommodate that need in your strategy for managing duplicate records.

29. MariaDB – SQL Injection Protection

The simple act of accepting user input opens the door to exploits. The problem stems primarily from the logical management of data, but luckily, it is fairly easy to avoid these major flaws.

Opportunities for SQL injection typically occur on users entering data like a name, and the code logic failing to analyze this input. The Code, instead, allows an attacker to insert a MariaDB statement, which will run on the database.

Always consider data entered by users, suspect and are in need of strong validation prior to any processing. Perform this validation through pattern matching. For example, if the expected input is a username, restrict entered characters to alphanumeric chars and underscores, and to a certain length. Review an example given below-

```
if (check_match("/^\w{8,20}$/", $_GET['user_name'], $matches))
{
   $result = mysql_query("SELECT * FROM system_users
                          WHERE user_name=$matches[0]");
}
 else
{
   echo "Invalid username";
}
```

Also, utilize the REGEXP operator and LIKE clauses in creating input constraints.

Consider all types of necessary explicit control of input such as-

- Control the escape characters used.
- Control the specific appropriate data types for input. Limit input to the necessary data type and size.
- Control the syntax of entered data. Do not allow anything outside of the needed pattern.
- Control the terms permitted. Blacklist SQL keywords.

You may not know the dangers of injection attacks, or may consider them insignificant, but they top the list of security concerns. Furthermore, consider the effect of these two entries-

```
1=1
-or-
*
```

Code allowing either of those to be entered along with the right command may result in revealing all user data on the database or deleting all data on the database, and neither injection is particularly clever. In some cases, attackers do not even spend time examining holes; they perform blind attacks with simple input.

Also, consider the pattern matching and regular expression tools provided by any programming/scripting language paired with MariaDB, which provide more control, and sometimes better control.

30. MariaDB – Backup Methods

Data serves as the foundation of business and operations, and with various possible threats (e.g., attackers, system failures, bad upgrades, and maintenance errors) out there, backups remain critical. These backups take many forms, and many options exist for creating them with an even wider set of options within those processes. The important things to remember are the database type, the critical information, and the structure involved. This information determines your best option.

OPTIONS

The main options for backups include logical backups and physical backups. Logical backups hold SQL statements for restoring data. Physical backups contain copies of data.

- **Logical backups** offer the flexibility of restoring data on another machine with a different configuration in contrast to physical backups, which are often limited to the same machine and database type. Logical backups occur at database and table level, and physical occur at directory and file level.

- **Physical backups** are smaller in size than logical, and also take less time to perform and restore. Physical backups also include log and configuration files, but logical backups do not.

Backup Tools

The main tool used for MariaDB backups is **mysqldump**. It offers logical backups and flexibility. It also proves an excellent option for small databases. **Mysqldump** dumps data into SQL, CSV, XML, and many other formats. Its output does not retain stored procedures, views, and events without explicit instruction.

There are three options for **mysqldump** backups-

- **Raw data** – Dump a table as a raw data file through the `--tab option`, which also specifies the destination of the file-

```
$ mysqldump -u root -p --no-create-info \
            --tab=/tmp PRODUCTS products_tbl
```

- **Data/Definitions export** – This option allows a single or multiple tables to be exported to a file, and supports backing up all existing databases on the host machine. Examine an example of exporting contents or definitions to a file-

```
$ mysqldump -u root -p PRODUCTS products_tbl > export_file.txt
```

- **Transfer –** You can also output databases and tables to another host-

```
$ mysqldump -u root -p database_name \
       | mysql -h other-host.com database_name
```

Using THE SELECT...INTO OUTFILE Statement

Another option for exporting data employs the SELECT...INTO OUTFILE statement. This simple option outputs the table into a simple formatted text file-

```
mysql> SELECT * FROM products_tbl
    -> INTO OUTFILE '/tmp/products.txt';
```

Its attributes allow formatting the file to your preferred specifications.

Note the following qualities of this statement-

- The file name must specify your desired location for the output.
- You need MariaDB file privileges to execute the statement.
- The output file name must be unique.
- You need login credentials on the host.
- In a UNIX environment, the output file is world readable, but its server ownership affects your ability to delete it. Ensure you have privileges.

Using CONNECT in Backups

The CONNECT handler allows exporting of data. This proves useful primarily in situations when the `SELECT...INTO OUTFILE` operation does not support the file format.

Review the following example-

```
create table products
engine=CONNECT table_type=XML file_name='products.htm' header=yes
option_list='name=TABLE,coltype=HTML,attribute=border=1;cellpadding=5'


select plugin_name handler, plugin_version version, plugin_author
author, plugin_description description, plugin_maturity maturity
from information_schema.plugins where plugin_type = 'STORAGE ENGINE';
```

Other Tools

Other options for backups are as follows-

- **XtraBackup** – This option targets XtraDB/InnoDB databases and works with any storage engine. Learn more about this tool from Percona's official site.

- **Snapshots** – Some filesystems allow snapshots. The process consists of flushing the tables with read lock, mounting the snapshot, unlocking the tables, copying the snapshot, and then unmounting the snapshot.

- **LVM** – This popular method employs a Perl script. It gets a read lock on every table and flushes caches to disk. Then it gets a snapshot and unlocks the tables. Consult the official **mylvmbackup** website for more information.

- **TokuBackup** – This solution provided by Percona provides hot backups taking into account the problems and limitations of InnoDB backup options. It produces a transactional sound copy of files while applications continue to manipulate them. Consult the Percona website for more information.

INNODB Considerations

InnoDB uses a buffer pool for performance enhancement. In a backup, configure InnoDB to avoid copying an entire table into the buffer pool because logical backups typically perform full table scans.

31. MariaDB – Backup Loading Methods

In this chapter, we will learn about various backup loading methods. Restoring a database from a backup is a simple and sometimes terribly long process.

There are three options in loading data: the LOAD DATA statement, mysqlimport, and a simple mysqldump restore.

Using LOAD DATA

The LOAD DATA statement functions as a bulk loader. Review an example of its use that loads a text file-

```
mysql> LOAD DATA LOCAL INFILE 'products_copy.txt' INTO TABLE empty_tbl;
```

Note the following qualities of a LOAD DATA statement-

- Use the LOCAL keyword to prevent MariaDB from performing a deep search of the host, and use a very specific path.
- The statement assumes a format consisting of lines terminated by linefeeds (newlines) and data values separated by tabs.
- Use the FIELDS clause to explicitly specify formatting of fields on a line. Use the LINES clause to specify line ending. Review an example below-

```
mysql> LOAD DATA LOCAL INFILE 'products_copy.txt' INTO TABLE empty_tbl
   FIELDS TERMINATED BY '|'
   LINES TERMINATED BY '\n';
```

- The statement assumes columns within the datafile use the same order of the table. If you need to set a different order, you can load the file as follows-

```
mysql> LOAD DATA LOCAL INFILE 'products_copy.txt'
     INTO TABLE empty_tbl (c, b, a);
```

Using MYSQLIMPORT

The mysqlimport tool acts as a LOAD DATA wrapper allowing the same operations from the command line.

Load data as follows-

```
$ mysqlimport -u root -p --local database_name source_file.txt
```

Specify formatting as follows-

```
$ mysqlimport -u root -p --local --fields-terminated-by="|" \
   --lines-terminated-by="\n"  database_name source_file.txt
```

Use the –-**columns** option to specify column order-

```
$ mysqlimport -u root -p --local --columns=c,b,a \
    database_name source_file.txt
```

Using MYSQLDUMP

Restoring with **mysqldump** requires this simple statement for loading the dump file back into the host-

```
shell> mysql database_name < source_file.sql
```

SPECIAL CHARACTERS AND QUOTES

In a LOAD DATA statement, quotes and special characters may not be interpreted correctly. The statement assumes unquoted values and treats backslashes as escape characters. Use the FIELDS clause to specify formatting. Point to quotes with "ENCLOSED BY," which causes the stripping of quotes from data values. Change escapes with "ESCAPED BY."

32. MariaDB – Useful Functions

This chapter contains a list of the most frequently used functions, offering definitions, explanations, and examples.

MariaDB Aggregate Functions

Most frequently used aggregate functions are given below-

Name	Description
COUNT	It counts the number of records. **Example**: `SELECT COUNT(*) FROM customer_table;`
MIN	It reveals the minimum value of a set of records. **Example**: `SELECT organization, MIN(account) FROM contracts GROUP BY organization;`
MAX	It reveals the maximum value of a set of records. **Example**: `SELECT organization, MAX(account_size) FROM contracts GROUP BY organization;`
AVG	It calculates the average value of a set of records. **Example**: `SELECT AVG(account_size) FROM contracts;`
SUM	It calculates the sum of a set of records. **Example**: `SELECT SUM(account_size) FROM contracts;`

MariaDB Age Calculation

The **TIMESTAMPDIFF** function provides a way to calculate age-

```
SELECT CURDATE() AS today;
SELECT ID, DOB, TIMESTAMPDIFF(YEAR,DOB,'2015-07-01') AS age
  FROM officer_info;
```

MariaDB String Concatenation

The **CONCAT** function returns the resulting string after a concatenation operation. You can utilize one or more arguments. Review its syntax given below-

```
SELECT CONCAT(item, item,...);
```

Review thefollowing example-

```
SELECT CONCAT('Ram', 'bu', 'tan');
Output:Rambutan
```

MariaDB Date/Time Functions

Given below are important date functions-

Name	Description
CURDATE()	It returns the date in yyyy-mm-dd or yyyymmdd format. **Example**: `SELECT CURDATE();`
DATE()	It returns the date in multiple formats. **Example**: `CREATE TABLE product_release_tbl (x DATE);`
CURTIME()	It returns the time in HH:MM:SS or HHMMSS.uuuuuu format. **Example**: `SELECT CURTIME();`
DATE_SUB()	It adds or subtracts a number of days from the specified date. **Example**: `SELECT DATE_SUB('2016-02-08', INTERVAL 60 DAY);`
DATEDIFF()	It determines the days between two dates. **Example**: `SELECT DATEDIFF('2016-01-01 23:59:59','2016-01-03');`
DATE ADD()	It adds or subtracts any unit of time to/from the date and time. **Example**: `SELECT DATE_ADD('2016-01-04 23:59:59', INTERVAL 22 SECOND);`
EXTRACT()	It extracts a unit from the date. **Example**: `SELECT EXTRACT(YEAR FROM '2016-01-08');`

NOW()	It returns the current date and time in either yyyy-mm-dd hh:mm:ss or yyyymmddhhmmss.uuuuuu format. **Example**: `SELECT NOW();`
DATE FORMAT()	It formats the date in accordance with the specified format string. **Example**: `SELECT DATE_FORMAT('2016-01-09 20:20:00', '%W %M %Y');`

Following are some important time functions-

Name	Description
HOUR()	It returns the hour of the time, or the hours elapsed. **Example**: `SELECT HOUR('19:17:09');`
LOCALTIME()	It functions exactly like NOW().
MICROSECOND()	It returns the microseconds of the time. **Example**: `SELECT MICROSECOND('16:30:00.543876');`
MINUTE()	It returns the minutes of the time. **Example**: `SELECT MINUTE('2016-05-22 17:22:01');`
SECOND()	It returns the seconds of the date. **Example**: `SELECT SECOND('2016-03-12 16:30:04.000001');`
TIME_FORMAT()	It formats the time in accordance with the specified format string. **Example**: `SELECT TIME_FORMAT('22:02:20', '%H %k %h %I %l');`
TIMESTAMP()	It provides a timestamp for an activity in the format yyyy-mm-dd hh:mm:dd. **Example**: `CREATE TABLE orders_ (ID INT, tmst TIMESTAMP);`

MariaDB Numeric Functions

Given below are some important numeric functions in MariaDB-

Name	Description
TRUNCATE()	It returns a truncated number to decimal place specification. **Example**: `SELECT TRUNCATE(101.222, 1);`
COS()	It returns the cosine of x radians. **Example**: `SELECT COS(PI());`
CEILING()	It returns the smallest integer not below x. **Example**: `SELECT CEILING(2.11);`
DEGREES()	It converts radians to degrees. **Example**: `SELECT DEGREES(PI());`
DIV()	It performs integer division. **Example**: `SELECT 100 DIV 4;`
EXP()	It returns e to the power of x. **Example**: `SELECT EXP(2);`
FLOOR()	It returns the largest integer not above x. **Example**: `SELECT FLOOR(2.01);`
LN()	It returns the natural logarithm of x. **Example**: `SELECT LN(3);`
LOG()	It returns the natural logarithm or the logarithm to a given base. **Example**: `SELECT LOG(3);`
SQRT()	It returns the square root. **Example**: `SELECT SQRT(16);`

MariaDB String Functions

Important string functions are given below-

Name	Description
INSTR()	It returns the position of the first instance of a substring. **Example**: `SELECT INSTR('rambutan', 'tan');`
RIGHT()	It returns the rightmost string characters. **Example**: `SELECT RIGHT('rambutan', 3);`
LENGTH()	It returns the byte length of a string. **Example**: `SELECT LENGTH('rambutan');`
LOCATE()	It returns the position of the first instance of a substring. **Example**: `SELECT LOCATE('tan', 'rambutan');`
INSERT()	It returns a string, with a specified substring at a certain position, that was modified. **Example**: `SELECT INSERT('ramputan', 4, 1, 'b');`
LEFT()	It returns the leftmost characters. **Example**: `SELECT LEFT('rambutan', 3);`
UPPER()	It changes characters to uppercase. **Example**: `SELECT UPPER(lastname);`
LOWER()	It changes characters to lowercase. **Example**: `SELECT LOWER(lastname);`
STRCMP()	It compares strings and returns 0 when they are equal. **Example**: `SELECT STRCMP('egg', 'cheese');`
REPLACE()	It returns a string after replacing characters. **Example**: `SELECT REPLACE('sully', 'l', 'n');`
REVERSE()	It reverses characters in a string. **Example**: `SELECT REVERSE('racecar');`
REPEAT()	It returns a string repeating given characters x times. **Example**: `SELECT REPEAT('ha ', 10);`

SUBSTRING()	It returns a substring from a string, starting at position x. **Example**: `SELECT SUBSTRING('rambutan',3);`
TRIM()	It removes trailing/leading characters from a string. **Example**: `SELECT TRIM(LEADING '_' FROM '_rambutan');`