MariaDB and SQL

SQL

Installing MariaDB

What is MariaDB

MariaDB is a database management system. It is one of the most popular ones in the world, and is free / open source! It is based on and built by some of the team from MySQL: a wildly successful and popular DBMS.

Because of its heritage, most tutorials and documentation from its older relative are compatible with MariaDB. This means the thousands upon thousands of documents, web pages, video tutorials, and more already out there can still help you out!

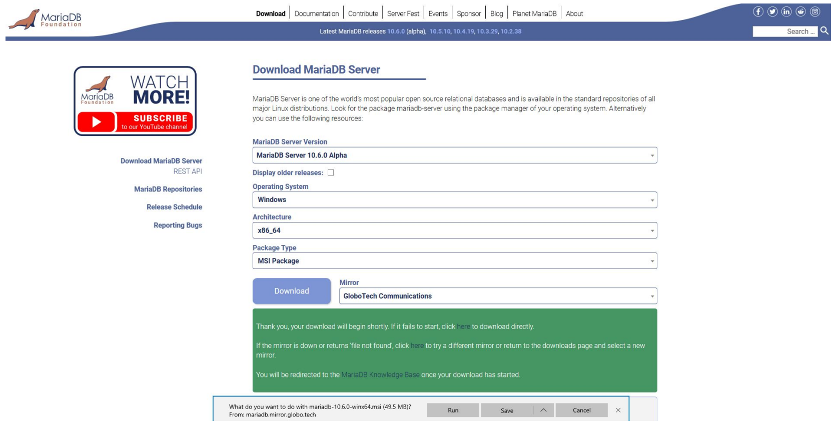
If you'd like to learn more about MariaDB, check out their website.

MariaDB Foundation

Download MariaDB

Navigate to the “Download” page on the MariaDB website. Here you can choose between different versions of the software, and download an installer.

Once downloaded, run the installer program to begin installation.



Latest MariaDB releases 10.6.0 (alpha), 10.5.10, 10.4.19, 10.3.29, 10.2.38

Download MariaDB Server

MariaDB Server is one of the world's most popular open source relational databases and is available in the standard repositories of all major Linux distributions. Look for the package mariadb-server using the package manager of your operating system. Alternatively you can use the following resources:

MariaDB Server Version: MariaDB Server 10.6.0 Alpha

Display older releases: ☐

Operating System: Windows

Architecture: x86_64

Package Type: MSI Package

Mirror: GloboTech Communications

Thank you, your download will begin shortly. If it fails to start, click here to download directly.

If the mirror is down or returns 'file not found', click here to try a different mirror or return to the downloads page and select a new mirror.

You will be redirected to the MariaDB Knowledge Base once your download has started.

What do you want to do with mariadb-10.6.0-winx64.msi (49.5 MB)?
From: mariadb.mirror.globo.tech

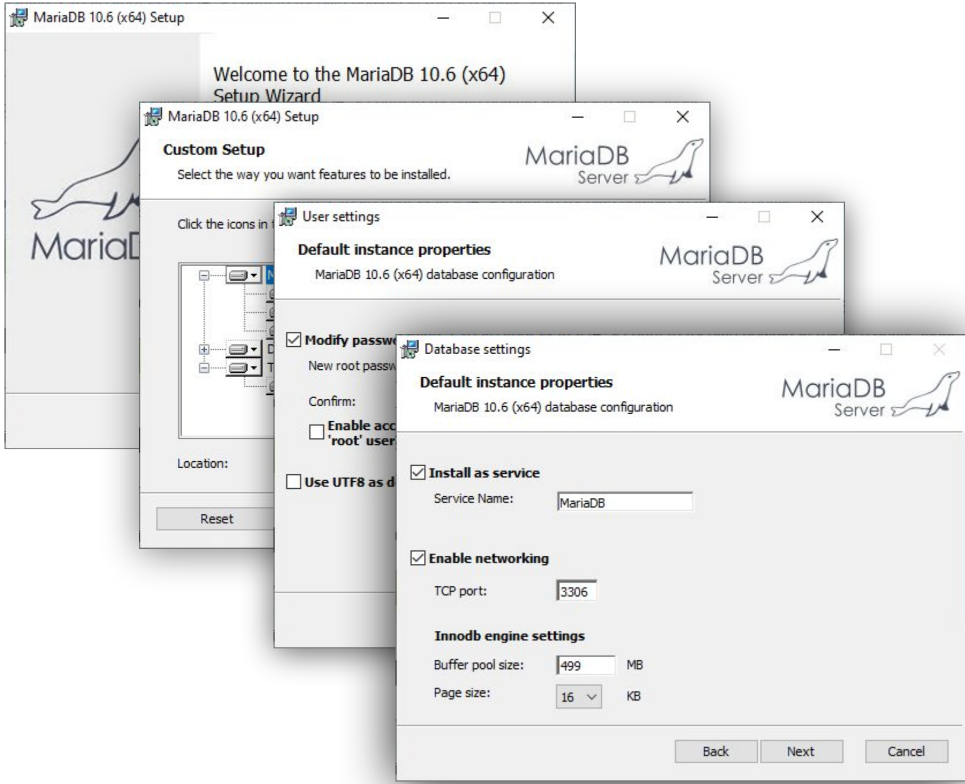
Install MariaDB

Follow the steps offered by the installer program.

Ensure you keep track of your database's root password, as you'll need this later in order to connect to it!

Using the default "TCP port" is a good idea if you do not already have a database software installed on your computer that might conflict. Using the default will mean any tutorials or programs will usually match your configuration, so less fiddling, customization, and overriding will be necessary during your time working on databases.

We recommend letting it install HeidiSQL, as this software is great for quickly reading your DB!



MariaDB 10.6 (x64) Setup

Welcome to the MariaDB 10.6 (x64) Setup Wizard

MariaDB 10.6 (x64) Setup

Custom Setup

Select the way you want features to be installed.

Click the icons in

Location:

Reset

User settings

Default instance properties

MariaDB 10.6 (x64) database configuration

☑ Modify passw

New root passw

Confirm:

☐ Enable acc 'root' user

☐ Use UTF8 as d

Database settings

Default instance properties

MariaDB 10.6 (x64) database configuration

☑ Install as service

Service Name: MariaDB

☑ Enable networking

TCP port: 3306

Innodb engine settings

Buffer pool size: 499 MB

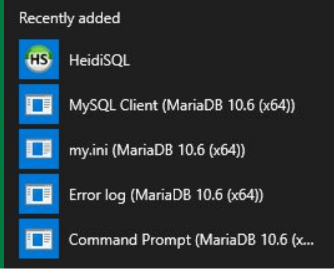
Page size: 16 KB

Before we Dive in

The MariaDB website has some great links and information that can help both beginners and veterans alike in their database journeys! To get caught up on both what we talked about previously, and prepare for what's to come when working with MariaDB, check out their article: Introduction to Relational Databases.

They'll explore with you the topic in respect to what MariaDB is capable of.

What was Installed


Recently added
HeidiSQL
MySQL Client (MariaDB 10.6 (x64))
my.ini (MariaDB 10.6 (x64))
Error log (MariaDB 10.6 (x64))
Command Prompt (MariaDB 10.6 (x...

Included in the default installation are a few new programs. One that was mentioned earlier is “HeidiSQL.” This program is a GUI (graphical user interface) way of reading and interacting with your SQL database(s). Keep this one in mind, as it will be a nice easy way of checking our work and writing queries.

Also included is the “MySQL Client.” This is the CLI (command-line interface) program for reading and interacting with your MariaDB database(s).

If you run into trouble, you may need to have a look at the “Error log” at some point. As MariaDB runs, if errors are encountered, messages describing what went wrong will be written to the log file.

Finally, if you’d like to make customizations to settings or the MariaDB configuration as a whole, the “my.ini” file is available and may be read and edited for this purpose. It is recommended you backup the “my.ini” file before making any changes, in case a new configuration causes errors and the changes need to be reverted. Edit this file carefully, and at your own risk!

SQL (Structured Query Language)

About SQL

SQL is a standard database language used to create, maintain, and retrieve information from a relational database.

Good to know:

- SQL is case-insensitive but it is recommended to type keywords in UPPERCASE (SELECT, JOIN, UPDATE, CREATE etc.) and things like table and column names in lowercase
- Each statement ends in a semicolon
- Comments are made using a double hyphen (--) at the beginning of a line (a commented line will not be executed as code, this is used to leave notes)
- MySQL is a slightly different flavour than MSSQL and queries written for one might not work for the other. We'll be focussing on MySQL during this course

Getting Started with SQL and MariaDB

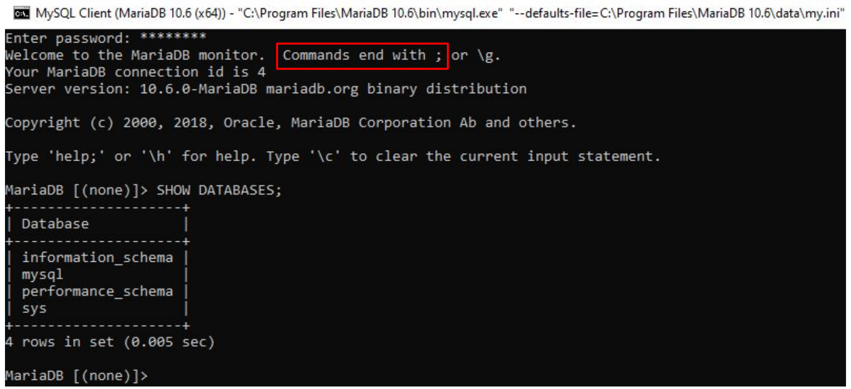
Open the “MySQL Client” program. It will prompt you for your password, so ensure you have it handy.

Once you’ve used your password to validate yourself and gain access to the terminal, you’ll be able to enter commands and begin working with the program.

Let’s try running a simple SQL line:

SHOW DATABASES;

As the content of the line suggests, this will print for us a list of the databases we have available at present.



MySQL Client (MariaDB 10.6 (x64)) - "C:\Program Files\MariaDB 10.6\bin\mysql.exe" "--defaults-file=C:\Program Files\MariaDB 10.6\data\my.ini"

```
Enter password: ********
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 4
Server version: 10.6.0-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

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

MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.005 sec)

MariaDB [(none)]>
```

Notice that the SQL statement ends in a semi-colon. The MySQL Client will not execute a line of code until it sees the semi-colon. This allows us to write multiple lines before sending!

*If you have begun writing instructions, but want to dismiss them, press CTRL+C to cancel.

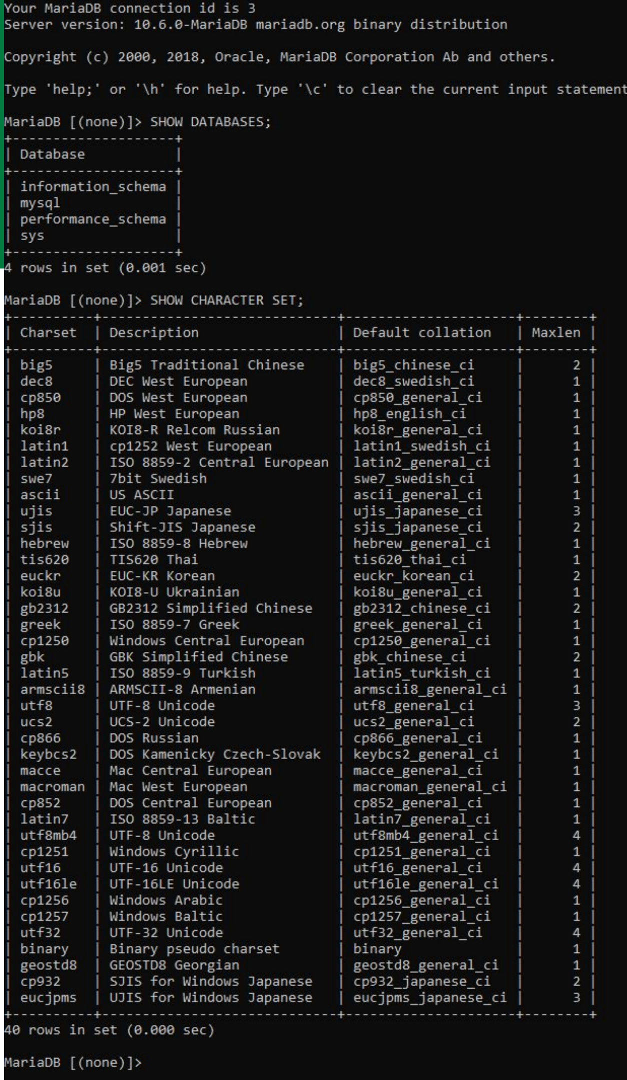
Character Sets

There are a wide array of character sets available for the data you store. Most popular in present day, especially as it pertains to the web, is “UTF-8 Unicode.” You may want to use different ones depending on the language or characters you plan to store.

Most previous versions of MySQL software use the “cp1252 West European” standard by default, which will likely be sufficient for most data you’ll want to store (UTF-8 is recommended for new databases.)

To see which character sets your version of MySQL has available, use the following command:

SHOW CHARACTER SET;



```
Your MariaDB connection id is 3
Server version: 10.6.0-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

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

MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.001 sec)

MariaDB [(none)]> SHOW CHARACTER SET;
+----------+-----------------------------+---------------------+--------+
| Charset  | Description                 | Default collation   | Maxlen |
+----------+-----------------------------+---------------------+--------+
| big5     | Big5 Traditional Chinese    | big5_chinese_ci     |      2 |
| dec8     | DEC West European           | dec8_swedish_ci     |      1 |
| cp850    | DOS West European           | cp850_general_ci    |      1 |
| hp8      | HP West European            | hp8_english_ci      |      1 |
| koi8r    | KOI8-R Relcom Russian       | koi8r_general_ci    |      1 |
| latin1   | cp1252 West European        | latin1_swedish_ci   |      1 |
| latin2   | ISO 8859-2 Central European | latin2_general_ci   |      1 |
| swe7     | 7bit Swedish                | swe7_swedish_ci     |      1 |
| ascii    | US ASCII                    | ascii_general_ci    |      1 |
| ujis     | EUC-JP Japanese             | ujis_japanese_ci    |      3 |
| sjis     | Shift-JIS Japanese          | sjis_japanese_ci    |      2 |
| hebrew   | ISO 8859-8 Hebrew           | hebrew_general_ci   |      1 |
| tis620   | TIS620 Thai                 | tis620_thai_ci      |      1 |
| euckr    | EUC-KR Korean               | euckr_korean_ci     |      2 |
| koi8u    | KOI8-U Ukrainian            | koi8u_general_ci    |      1 |
| gb2312   | GB2312 Simplified Chinese   | gb2312_chinese_ci   |      2 |
| greek    | ISO 8859-7 Greek            | greek_general_ci    |      1 |
| cp1250   | Windows Central European    | cp1250_general_ci   |      1 |
| gbk      | GBK Simplified Chinese      | gbk_chinese_ci      |      2 |
| latin5   | ISO 8859-9 Turkish          | latin5_turkish_ci   |      1 |
| armscii8 | ARMSCII-8 Armenian          | armscii8_general_ci |      1 |
| utf8     | UTF-8 Unicode               | utf8_general_ci     |      3 |
| ucs2     | UCS-2 Unicode               | ucs2_general_ci     |      2 |
| cp866    | DOS Russian                 | cp866_general_ci    |      1 |
| keybcs2  | DOS Kamenicky Czech-Slovak  | keybcs2_general_ci  |      1 |
| macce    | Mac Central European        | macce_general_ci    |      1 |
| macroman | Mac West European           | macroman_general_ci |      1 |
| cp852    | DOS Central European        | cp852_general_ci    |      1 |
| latin7   | ISO 8859-13 Baltic          | latin7_general_ci   |      1 |
| utf8mb4  | UTF-8 Unicode               | utf8mb4_general_ci  |      4 |
| cp1251   | Windows Cyrillic            | cp1251_general_ci   |      1 |
| utf16    | UTF-16 Unicode              | utf16_general_ci    |      4 |
| utf16le  | UTF-16LE Unicode            | utf16le_general_ci  |      4 |
| cp1256   | Windows Arabic              | cp1256_general_ci   |      1 |
| cp1257   | Windows Baltic              | cp1257_general_ci   |      1 |
| utf32    | UTF-32 Unicode              | utf32_general_ci    |      4 |
| binary   | Binary pseudo charset       | binary              |      1 |
| geostd8  | GEOSTD8 Georgian            | geostd8_general_ci  |      1 |
| cp932    | SJIS for Windows Japanese   | cp932_japanese_ci   |      2 |
| eucjpms  | UJIS for Windows Japanese   | eucjpms_japanese_ci |      3 |
+----------+-----------------------------+---------------------+--------+
40 rows in set (0.000 sec)

MariaDB [(none)]>
```

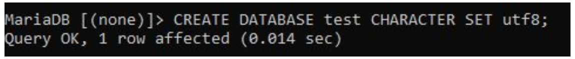
Working with Databases

Creating and Dropping Databases

Let's try creating a new database, and then deleting it. Be careful not to target the default databases, or anything else that you may affect by accident. Let's give it a shot!

First, let's try creating a database called "test":

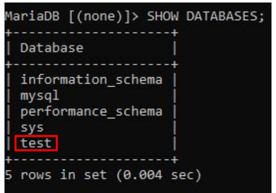
CREATE DATABASE test CHARACTER SET utf8;



```
MariaDB [(none)]> CREATE DATABASE test CHARACTER SET utf8;
Query OK, 1 row affected (0.014 sec)
```

Success! Let's check our full database list:

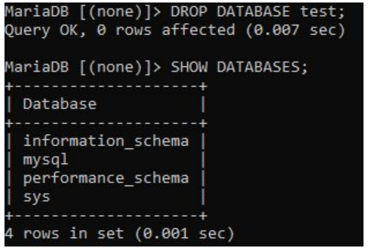
SHOW DATABASES;



```
MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| test               |
+--------------------+
5 rows in set (0.004 sec)
```

To delete a database, you may enter the DROP keyword followed by the target database name:

DROP DATABASE test;



```
MariaDB [(none)]> DROP DATABASE test;
Query OK, 0 rows affected (0.007 sec)

MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.001 sec)
```

And just like that—our database is gone! Be *very*, ***very*** careful with commands like this. What if you had just spent hours on a database!?

Basic Operations and Structure

Core Data Operations

When managing data, there are four critical base operations that are usually available (referred to, typically, as CRUD):

- **Create**
  Add a new entry of this data.
- **Read**
  View an existing entry of this data.
- **Update**
  Edit an existing entry of this data.
- **Delete**
  Remove or destroy an existing entry of this data.

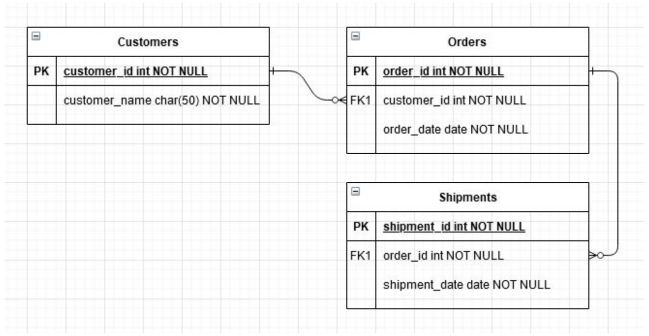
You'll notice when working with the database itself, we've already read the list of databases, created a database, and deleted one!

Database Structure

Recall how databases are structured:

- Each database that is created should have one or more tables.
- Each table should have one or more columns describing what data will be stored in each entry.
- Each table should have zero or more rows, representing each entry or record in that table.

As an example, let's take a look at the default (at time of writing) database ERD from Draw.IO:

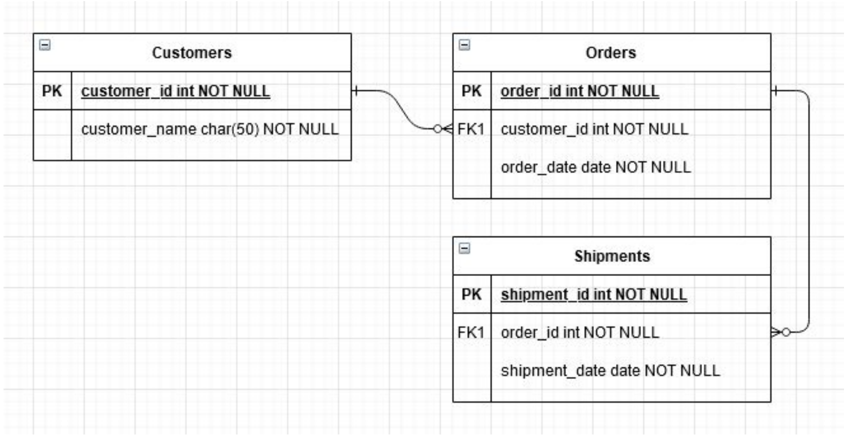


Breaking Down the Example

This database, let's call it "store," has three tables: Customers, Orders, and Shipments.

Each table represents a different type of data, which may require specific columns to represent. Notice the distinct difference between each table in terms of naming and data types for each column.

Let's see about creating a "store" database, setting up a "Customers" table, its columns, and adding some records.



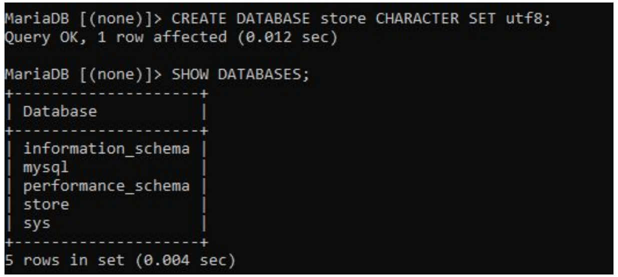
Create the Database

Alright, just like we did before, let's create a database. This time we'll name it "store":

CREATE DATABASE store CHARACTER SET utf8;

Confirm that it was added to your list of databases:

SHOW DATABASES;



```
MariaDB [(none)]> CREATE DATABASE store CHARACTER SET utf8;
Query OK, 1 row affected (0.012 sec)

MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| store              |
| sys                |
+--------------------+
5 rows in set (0.004 sec)
```

Create a Table

Prepare to Create a Table

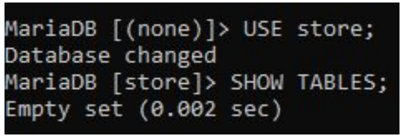
Before we can add tables, we must first tell MySQL which database we'd like to use:

USE store;

Next, we should check which tables are available in the chosen database:

SHOW TABLES;

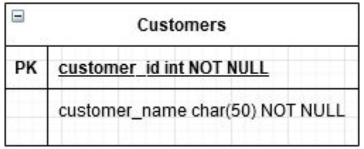
There should be nothing so far, as we haven't created any just yet!



```
MariaDB [(none)]> USE store;
Database changed
MariaDB [store]> SHOW TABLES;
Empty set (0.002 sec)
```

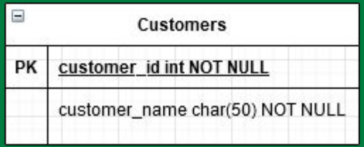
When creating a table, you'll need to do a little more planning and thinking, as this is where you'll also want to define the columns that will exist.

What we see in the ERD suggests we'll need an INT (whole number) value representing a primary key that auto-increments, as well as a text (CHAR) value of up to 50 characters that is not allowed to be empty (NULL.)



Customers	
PK	customer_id int NOT NULL
	customer_name char(50) NOT NULL

Create the Table

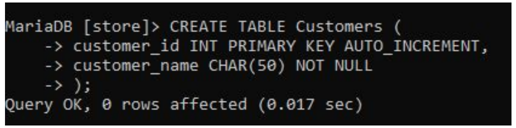


	Customers
PK	customer_id int NOT NULL
	customer_name char(50) NOT NULL

Pay close attention to the syntax we use to accomplish this in the following SQL. Don't forget your data types!

```
CREATE TABLE Customers (
  customer_id INT PRIMARY KEY AUTO_INCREMENT,
  customer_name CHAR(50) NOT NULL
 );
```

This will create the table and define the two desired columns.



```
MariaDB [store]> CREATE TABLE Customers (
    -> customer_id INT PRIMARY KEY AUTO_INCREMENT,
    -> customer_name CHAR(50) NOT NULL
    -> );
Query OK, 0 rows affected (0.017 sec)
```

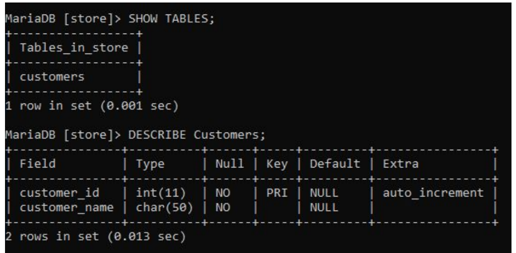
Once this has been executed, you can check the tables list to see if it has been added:

```
SHOW TABLES;
```

Ah, there it is! Now lets check to see how our columns are looking:

```
DESCRIBE Customers;
```

Excellent, everything is as we'd hoped!



```
MariaDB [store]> SHOW TABLES;
+-----------------+
| Tables_in_store |
+-----------------+
| customers       |
+-----------------+
1 row in set (0.001 sec)

MariaDB [store]> DESCRIBE Customers;
+---------------+----------+------+-----+---------+----------------+
| Field         | Type     | Null | Key | Default | Extra          |
+---------------+----------+------+-----+---------+----------------+
| customer_id   | int(11)  | NO   | PRI | NULL    | auto_increment |
| customer_name | char(50) | NO   |     | NULL    |                |
+---------------+----------+------+-----+---------+----------------+
2 rows in set (0.013 sec)
```

About AUTO_INCREMENT

- Generates a unique identity for each row/record
- Only works with integer data types. That includes smallint, mediumint, int, and bigint
- If you have records with 1, 2, 3, & 4 as IDs and you delete the record with an ID of 2 the next record you add will have an ID of 5. AUTO_INCREMENT doesn’t backfill (Which is why deleting records isn’t best practice, setting a flag is. E.g. isActive)
- Increments the last inserted ID by 1. If you insert a row with an ID of 100, the next row will have an ID of 101
- Official Documentation

Breaking Down the customer_id Column

Let’s have a closer look at our `customer_id` column definition:

```
customer_id INT PRIMARY KEY AUTO_INCREMENT
```

In the above line we:

- Named it `customer_id`
- Set its data type to INT (integer)
- Let the database know that this is a PRIMARY KEY (that is to say, this column is used as the main way to identify specific records)
- Let the database know to automatically increment the integer with each new row

It is important to note that “PRIMARY KEY” is what is called a constraint. Let’s have a closer look at constraints.

Constraints

Constraints

When we'd like to assign rules or designate importance to our tables and their columns, we may use constraints. Review the following table for the options MariaDB affords us:

PRIMARY KEY	The primary key is used for identifying or targeting specific rows in a table. When a table has a primary key, using it will result in more efficient querying.
FOREIGN KEY	Foreign keys are meant to match with a primary key in *another*, separate table. For instance: if each customer in Customers had an order, you might put the order_id here so that you know which order is associated with them.
UNIQUE	Only unique values that do not already appear in this table, in this column, may be entered in any given row / record.
CHECK	Checks that data entered into this column follows a specific rule or meets a given condition.

Checks

Check constraints are of particular importance, in that we can use them to have records only editable or inserted when a requirement of our own design is met. For example, say we had a product table, and the price would need to always be greater than zero—our store would never have a free or trial product.

In such a case, when making the table, we may have a constraint that functions like so:

```
CREATE TABLE Products (
   product_id INT PRIMARY KEY UNIQUE AUTO_INCREMENT,
   product_name CHAR(50) NOT NULL,
   product_price DECIMAL(20, 2) ZEROFILL CHECK(product_price>0.00) NOT NULL
 );
```

It is CHECK that will test each potential new row, or row value. Notice how it will compare the `product_price` column with the value "0.00" to see which is higher. If the check rings true (if product price is greater than zero) the new entry will be allowed, otherwise the row will not be added/modified.

Changing Table Structure

Making Changes to a Table’s Structure

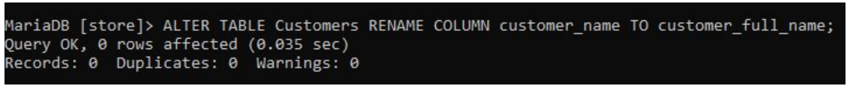
If a change to a table needs to be made, we are afforded the opportunity to ALTER it. This does not only apply to columns, but let’s first look at how we might add, update, or remove a column in our table.

Note that you should only make changes prior to using or filling a table with data. It can be risky to change column names, data types, rules, constraints, etc. after the table has been populated. Planning goes a long ways when it comes to database design, make changes before you insert records. This way, your records will be stable and you will not have frequent issues with unintended results or errors.

Rename a Table Column

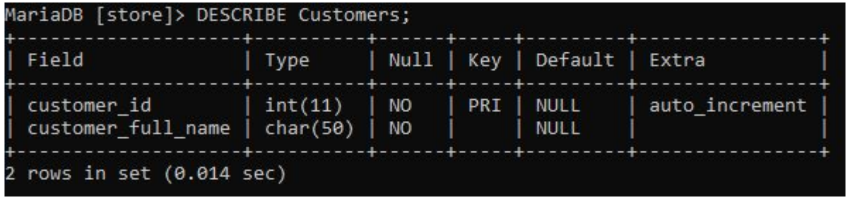
Firstly, let's practice with a renaming of our table column:

ALTER TABLE Customers
  RENAME COLUMN customer_name TO customer_full_name;



```
MariaDB [store]> ALTER TABLE Customers RENAME COLUMN customer_name TO customer_full_name;
Query OK, 0 rows affected (0.035 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

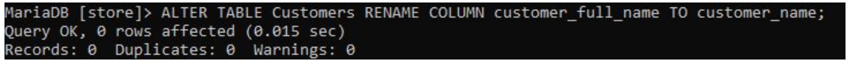
To confirm the change, let's ask our database to describe the table so we can review its columns:



```
MariaDB [store]> DESCRIBE Customers;
+--------------------+----------+------+-----+---------+----------------+
| Field              | Type     | Null | Key | Default | Extra          |
+--------------------+----------+------+-----+---------+----------------+
| customer_id        | int(11)  | NO   | PRI | NULL    | auto_increment |
| customer_full_name | char(50) | NO   |     | NULL    |                |
+--------------------+----------+------+-----+---------+----------------+
2 rows in set (0.014 sec)
```

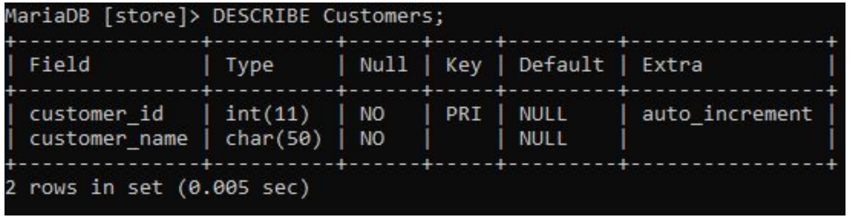
This change in name doesn't really add to the description of our column in this case—so let's practice one more time by changing it back:

ALTER TABLE Customers
  RENAME COLUMN customer_name TO customer_full_name;



```
MariaDB [store]> ALTER TABLE Customers RENAME COLUMN customer_full_name TO customer_name;
Query OK, 0 rows affected (0.015 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

And again, we'll confirm the change:



```
MariaDB [store]> DESCRIBE Customers;
+---------------+----------+------+-----+---------+----------------+
| Field         | Type     | Null | Key | Default | Extra          |
+---------------+----------+------+-----+---------+----------------+
| customer_id   | int(11)  | NO   | PRI | NULL    | auto_increment |
| customer_name | char(50) | NO   |     | NULL    |                |
+---------------+----------+------+-----+---------+----------------+
2 rows in set (0.005 sec)
```

Adding a new Column

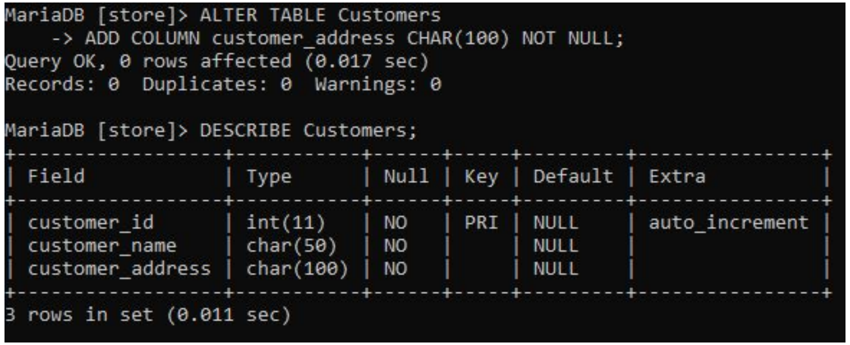
To add new columns after-the-fact, you can use the ADD COLUMN keywords in an ALTER TABLE statement. Let's say we wanted to add an address column:

```
ALTER TABLE Customers
  ADD COLUMN customer_address CHAR(100) NOT NULL;
```

Confirm your addition is in place:

```
DESCRIBE Customers;
```

Perfect; there it is!



```
MariaDB [store]> ALTER TABLE Customers
    -> ADD COLUMN customer_address CHAR(100) NOT NULL;
Query OK, 0 rows affected (0.017 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [store]> DESCRIBE Customers;
+------------------+-----------+------+-----+---------+----------------+
| Field            | Type      | Null | Key | Default | Extra          |
+------------------+-----------+------+-----+---------+----------------+
| customer_id      | int(11)   | NO   | PRI | NULL    | auto_increment |
| customer_name    | char(50)  | NO   |     | NULL    |                |
| customer_address | char(100) | NO   |     | NULL    |                |
+------------------+-----------+------+-----+---------+----------------+
3 rows in set (0.011 sec)
```

Updating an Existing Column

We can update an existing column, changing its definition. This could include its data type, whether it is nullable, the permitted length of entries in the column, whether or not it auto increments, or any other part of a column definition.

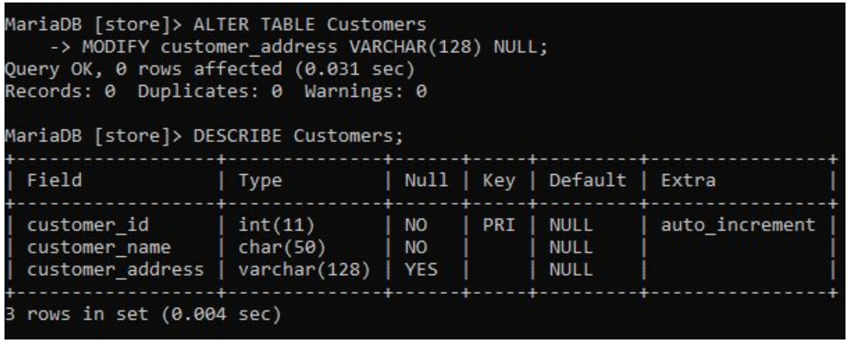
Let's give it a shot by changing the data type and length for our new address column:

```
ALTER TABLE Customers
  MODIFY customer_address VARCHAR(128) NULL;
```

*Beware making changes after data has been added to the table! This can damage, corrupt, or cause otherwise unintended consequences to your data and table stability!

Print the table description to ensure your column changes are reflected:

```
DESCRIBE Customers;
```



```
MariaDB [store]> ALTER TABLE Customers
    -> MODIFY customer_address VARCHAR(128) NULL;
Query OK, 0 rows affected (0.031 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [store]> DESCRIBE Customers;
+------------------+--------------+------+-----+---------+----------------+
| Field            | Type         | Null | Key | Default | Extra          |
+------------------+--------------+------+-----+---------+----------------+
| customer_id      | int(11)      | NO   | PRI | NULL    | auto_increment |
| customer_name    | char(50)     | NO   |     | NULL    |                |
| customer_address | varchar(128) | YES  |     | NULL    |                |
+------------------+--------------+------+-----+---------+----------------+
3 rows in set (0.004 sec)
```

Removing a Column

Perhaps we realize we don't need the address—it isn't necessary for our database or application. We'd want to remove it to avoid confusion!

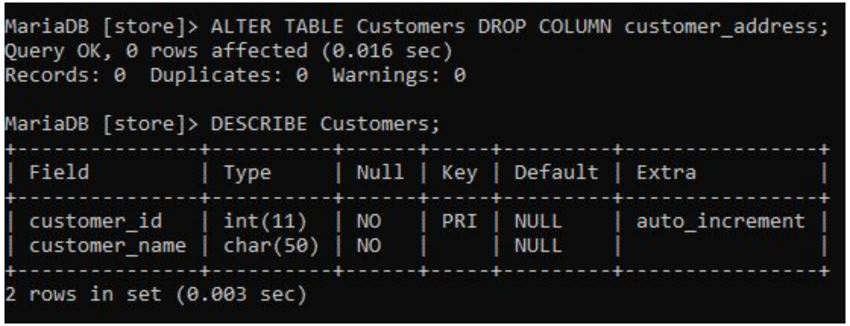
As with any deletion, use this with EXTREME caution! You don't want to lose all the information in an existing table unless you are **absolutely** certain that this is the desired outcome.

We can remove a column like so:

```
ALTER TABLE Customers
  DROP COLUMN customer_address;
```

Let's check the table structure to confirm:

```
DESCRIBE Customers;
```



```
MariaDB [store]> ALTER TABLE Customers DROP COLUMN customer_address;
Query OK, 0 rows affected (0.016 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [store]> DESCRIBE Customers;
+---------------+----------+------+-----+---------+----------------+
| Field         | Type     | Null | Key | Default | Extra          |
+---------------+----------+------+-----+---------+----------------+
| customer_id   | int(11)  | NO   | PRI | NULL    | auto_increment |
| customer_name | char(50) | NO   |     | NULL    |                |
+---------------+----------+------+-----+---------+----------------+
2 rows in set (0.003 sec)
```

Alright, we're back to the way things were!

Recommended Reading

It’s a good idea to read up more on what we’ve covered today. Have a look at the following:

- Beaulieu, A. (March 2020). Learning SQL, 3rd Edition. O’Reilly Media, Inc.
    - Chapter 2. Creating and Populating a Database
        - Creating a MySQL Database
        - Using the mysql Command-Line Tool
        - Table Creation