**ntroduction**

[MySQL](https://www.mysql.com/) is an open-source database management system, commonly installed as part of the popular [LAMP](https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu-16-04) (Linux, Apache, MySQL, PHP/Python/Perl) stack. It uses a relational database and SQL (Structured Query Language) to manage its data.

The short version of the installation is simple: update your package index, install the mysql-server package, and then run the included security script.

* sudo apt-get update
* sudo apt-get install mysql-server
* mysql\_secure\_installation

This tutorial will explain how to install MySQL version 5.7 on a Ubuntu 16.04 server. However, if you're looking to update an existing MySQL installation to version 5.7, you can read [this MySQL 5.7 update guide](https://www.digitalocean.com/community/tutorials/how-to-prepare-for-your-mysql-5-7-upgrade) instead.

**Prerequisites**

To follow this tutorial, you will need:

* One Ubuntu 16.04 server set up by following [this initial server setup guide](https://www.digitalocean.com/community/tutorials/initial-server-setup-with-ubuntu-16-04), including a sudo non-root user and a firewall.

**Step 1 — Installing MySQL**

On Ubuntu 16.04, only the latest version of MySQL is included in the APT package repository by default. At the time of writing, that's MySQL 5.7

To install it, simply update the package index on your server and install the default package with apt-get.

* sudo apt-get update
* sudo apt-get install mysql-server

You'll be prompted to create a root password during the installation. Choose a secure one and make sure you remember it, because you'll need it later. Next, we'll finish configuring MySQL.

**Step 2 — Configuring MySQL**

For fresh installations, you'll want to run the included security script. This changes some of the less secure default options for things like remote root logins and sample users. On older versions of MySQL, you needed to initialize the data directory manually as well, but this is done automatically now.

Run the security script.

* mysql\_secure\_installation

This will prompt you for the root password you created in Step 1. You can press Y and then ENTER to accept the defaults for all the subsequent questions, with the exception of the one that asks if you'd like to change the root password. You just set it in Step 1, so you don't have to change it now. For a more detailed walkthrough of these options, you can see [this step of the LAMP installation tutorial](https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu-16-04#step-2-install-mysql).

To initialize the MySQL data directory, you would use mysql\_install\_db for versions before 5.7.6, and mysqld --initialize for 5.7.6 and later. However, if you installed MySQL from the Debian distribution, like in Step 1, the data directory was initialized automatically; you don't have to do anything. If you try running the command anyway, you'll see the following error:

Output

2016-03-07T20:11:15.998193Z 0 [ERROR] --initialize specified but the data directory has files in it. Aborting.

Finally, let's test the MySQL installation.

**Step 3 — Testing MySQL**

Regardless of how you installed it, MySQL should have started running automatically. To test this, check its status.

* systemctl status mysql.service

You'll see output similar to the following:

Output

● mysql.service - MySQL Community Server

Loaded: loaded (/lib/systemd/system/mysql.service; enabled; vendor preset: en

Active: active (running) since Wed 2016-11-23 21:21:25 UTC; 30min ago

Main PID: 3754 (mysqld)

Tasks: 28

Memory: 142.3M

CPU: 1.994s

CGroup: /system.slice/mysql.service

└─3754 /usr/sbin/mysqld

If MySQL isn't running, you can start it with sudo systemctl start mysql.

For an additional check, you can try connecting to the database using the mysqladmin tool, which is a client that lets you run administrative commands. For example, this command says to connect to MySQL as **root** (-u root), prompt for a password (-p), and return the version.

* mysqladmin -p -u root version

You should see output similar to this:

Output

mysqladmin Ver 8.42 Distrib 5.7.16, for Linux on x86\_64

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its

affiliates. Other names may be trademarks of their respective

owners.

Server version 5.7.16-0ubuntu0.16.04.1

Protocol version 10

Connection Localhost via UNIX socket

UNIX socket /var/run/mysqld/mysqld.sock

Uptime: 30 min 54 sec

Threads: 1 Questions: 12 Slow queries: 0 Opens: 115 Flush tables: 1 Open tables: 34 Queries per second avg: 0.006

This means MySQL is up and running.

If the administrator creates your database for you when setting up your permissions, you can begin using it. Otherwise, you need to create it yourself:

mysql> CREATE DATABASE menagerie;

Under Unix, database names are case-sensitive (unlike SQL keywords), so you must always refer to your database as menagerie, not as Menagerie, MENAGERIE, or some other variant. This is also true for table names. (Under Windows, this restriction does not apply, although you must refer to databases and tables using the same lettercase throughout a given query. However, for a variety of reasons, the recommended best practice is always to use the same lettercase that was used when the database was created.)

Note

If you get an error such as ERROR 1044 (42000): Access denied for user 'micah'@'localhost' to database 'menagerie' when attempting to create a database, this means that your user account does not have the necessary privileges to do so. Discuss this with the administrator or see [Section 6.2, “The MySQL Access Privilege System”](https://dev.mysql.com/doc/refman/8.0/en/privilege-system.html).

Creating a database does not select it for use; you must do that explicitly. To make menagerie the current database, use this statement:

mysql> USE menagerie

Database changed

reating the database is the easy part, but at this point it is empty, as [SHOW TABLES](https://dev.mysql.com/doc/refman/5.5/en/show-tables.html) tells you:

mysql> SHOW TABLES;

Empty set (0.00 sec)

The harder part is deciding what the structure of your database should be: what tables you need and what columns should be in each of them.

You want a table that contains a record for each of your pets. This can be called the pet table, and it should contain, as a bare minimum, each animal's name. Because the name by itself is not very interesting, the table should contain other information. For example, if more than one person in your family keeps pets, you might want to list each animal's owner. You might also want to record some basic descriptive information such as species and sex.

How about age? That might be of interest, but it is not a good thing to store in a database. Age changes as time passes, which means you'd have to update your records often. Instead, it is better to store a fixed value such as date of birth. Then, whenever you need age, you can calculate it as the difference between the current date and the birth date. MySQL provides functions for doing date arithmetic, so this is not difficult. Storing birth date rather than age has other advantages, too:

* You can use the database for tasks such as generating reminders for upcoming pet birthdays. (If you think this type of query is somewhat silly, note that it is the same question you might ask in the context of a business database to identify clients to whom you need to send out birthday greetings in the current week or month, for that computer-assisted personal touch.)
* You can calculate age in relation to dates other than the current date. For example, if you store death date in the database, you can easily calculate how old a pet was when it died.

You can probably think of other types of information that would be useful in the pet table, but the ones identified so far are sufficient: name, owner, species, sex, birth, and death.

Use a [CREATE TABLE](https://dev.mysql.com/doc/refman/5.5/en/create-table.html) statement to specify the layout of your table:

mysql> CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),

-> species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);

[VARCHAR](https://dev.mysql.com/doc/refman/5.5/en/char.html) is a good choice for the name, owner, and species columns because the column values vary in length. The lengths in those column definitions need not all be the same, and need not be 20. You can normally pick any length from 1 to 65535, whatever seems most reasonable to you. If you make a poor choice and it turns out later that you need a longer field, MySQL provides an [ALTER TABLE](https://dev.mysql.com/doc/refman/5.5/en/alter-table.html) statement.

Several types of values can be chosen to represent sex in animal records, such as 'm' and 'f', or perhaps 'male' and 'female'. It is simplest to use the single characters 'm' and 'f'.

The use of the [DATE](https://dev.mysql.com/doc/refman/5.5/en/datetime.html) data type for the birth and death columns is a fairly obvious choice.

Once you have created a table, [SHOW TABLES](https://dev.mysql.com/doc/refman/5.5/en/show-tables.html) should produce some output:

mysql> SHOW TABLES;

+---------------------+

| Tables in menagerie |

+---------------------+

| pet |

+---------------------+

To verify that your table was created the way you expected, use a [DESCRIBE](https://dev.mysql.com/doc/refman/5.5/en/describe.html) statement:

mysql> DESCRIBE pet;

+---------+-------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+---------+-------------+------+-----+---------+-------+

| name | varchar(20) | YES | | NULL | |

| owner | varchar(20) | YES | | NULL | |

| species | varchar(20) | YES | | NULL | |

| sex | char(1) | YES | | NULL | |

| birth | date | YES | | NULL | |

| death | date | YES | | NULL | |

+---------+-------------+------+-----+---------+-------+

You can use [DESCRIBE](https://dev.mysql.com/doc/refman/5.5/en/describe.html) any time, for example, if you forget the names of the columns in your table or what types they have.

By default, **mysqldump** writes information as SQL statements to the standard output. You can save the output in a file:

shell> mysqldump [*arguments*] > *file\_name*

To dump all databases, invoke **mysqldump** with the [--all-databases](https://dev.mysql.com/doc/refman/5.7/en/mysqldump.html#option_mysqldump_all-databases) option:

shell> mysqldump --all-databases > dump.sql

To dump only specific databases, name them on the command line and use the [--databases](https://dev.mysql.com/doc/refman/5.7/en/mysqldump.html#option_mysqldump_databases) option:

shell> mysqldump --databases db1 db2 db3 > dump.sql

The [--databases](https://dev.mysql.com/doc/refman/5.7/en/mysqldump.html#option_mysqldump_databases) option causes all names on the command line to be treated as database names. Without this option, **mysqldump** treats the first name as a database name and those following as table names.

With [--all-databases](https://dev.mysql.com/doc/refman/5.7/en/mysqldump.html#option_mysqldump_all-databases) or [--databases](https://dev.mysql.com/doc/refman/5.7/en/mysqldump.html#option_mysqldump_databases), **mysqldump** writes [CREATE DATABASE](https://dev.mysql.com/doc/refman/5.7/en/create-database.html) and [USE](https://dev.mysql.com/doc/refman/5.7/en/use.html) statements prior to the dump output for each database. This ensures that when the dump file is reloaded, it creates each database if it does not exist and makes it the default database so database contents are loaded into the same database from which they came. If you want to cause the dump file to force a drop of each database before recreating it, use the [--add-drop-database](https://dev.mysql.com/doc/refman/5.7/en/mysqldump.html#option_mysqldump_add-drop-database) option as well. In this case, **mysqldump** writes a [DROP DATABASE](https://dev.mysql.com/doc/refman/5.7/en/drop-database.html) statement preceding each [CREATE DATABASE](https://dev.mysql.com/doc/refman/5.7/en/create-database.html) statement.

To dump a single database, name it on the command line:

shell> mysqldump --databases test > dump.sql

In the single-database case, it is permissible to omit the [--databases](https://dev.mysql.com/doc/refman/5.7/en/mysqldump.html#option_mysqldump_databases) option:

shell> mysqldump test > dump.sql

The difference between the two preceding commands is that without [--databases](https://dev.mysql.com/doc/refman/5.7/en/mysqldump.html#option_mysqldump_databases), the dump output contains no [CREATE DATABASE](https://dev.mysql.com/doc/refman/5.7/en/create-database.html) or [USE](https://dev.mysql.com/doc/refman/5.7/en/use.html) statements. This has several implications:

* When you reload the dump file, you must specify a default database name so that the server knows which database to reload.
* For reloading, you can specify a database name different from the original name, which enables you to reload the data into a different database.
* If the database to be reloaded does not exist, you must create it first.
* Because the output will contain no [CREATE DATABASE](https://dev.mysql.com/doc/refman/5.7/en/create-database.html) statement, the [--add-drop-database](https://dev.mysql.com/doc/refman/5.7/en/mysqldump.html#option_mysqldump_add-drop-database) option has no effect. If you use it, it produces no [DROP DATABASE](https://dev.mysql.com/doc/refman/5.7/en/drop-database.html) statement.

To dump only specific tables from a database, name them on the command line following the database name:

shell> mysqldump test t1 t3 t7 > dump.sql

**Create BackUp Of Data**

If it's an entire DB, then:

$ mysqldump -u [uname] -p[pass] db\_name > db\_backup.sql

If it's all DBs, then:

$ mysqldump -u [uname] -p[pass] --all-databases > all\_db\_backup.sql

If it's specific tables within a DB, then:

$ mysqldump -u [uname] -p[pass] db\_name table1 table2 > table\_backup.sql

You can even go as far as auto-compressing the output using gzip (if your DB is very big):

$ mysqldump -u [uname] -p[pass] db\_name | gzip > db\_backup.sql.gz

If you want to do this **remotely** and you have the access to the server in question, then the following would work (presuming the MySQL server is on port 3306):

$ mysqldump -P 3306 -h [ip\_address] -u [uname] -p[pass] db\_name > db\_backup.sql

It should drop the .sql file in the folder you run the command-line from.

**MYSQL Using Docker:**

#### Downloading a MySQL Server Docker Image

Downloading the server image in a separate step is not strictly necessary; however, performing this step before you create your Docker container ensures your local image is up to date. To download the MySQL Community Server image, run this command:

docker pull mysql/mysql-server:tag

The *tag* is the label for the image version you want to pull (for example, 5.5, 5.6, 5.7, 8.0, or latest). If **:*tag*** is omitted, the latest label is used, and the image for the latest GA version of MySQL Community Server is downloaded. Refer to the list of tags for available versions on the [mysql/mysql-server page in the Docker Hub](https://hub.docker.com/r/mysql/mysql-server/tags/).

You can list downloaded Docker images with this command:

shell> docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

mysql/mysql-server latest 3157d7f55f8d 4 weeks ago 241MB

#### Starting a MySQL Server Instance

Start a new Docker container for the MySQL Server with this command:

docker run --name=mysql1 -d mysql/mysql-server:tag

The --name option, for supplying a custom name for your server container (mysql1 in the example), is optional; if no container name is supplied, a random one is generated. If the Docker image of the specified name and tag has not been downloaded by an earlier **docker pull** or **docker run** command, the image is now downloaded. After download completes, initialization for the container begins, and the container appears in the list of running containers when you run the **docker ps** command; for example:

shell> docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

a24888f0d6f4 mysql/mysql-server "/entrypoint.sh my..." 14 seconds ago Up 13 seconds (health: starting) 3306/tcp, 33060/tcp mysql1

The container initialization might take some time. When the server is ready for use, the STATUS of the container in the output of the **docker ps** command changes from (health: starting) to (healthy).

The -d option used in the **docker run** command above makes the container run in the background. Use this command to monitor the output from the container:

docker logs mysql1

Once initialization is finished, the command's output is going to contain the random password generated for the root user; check the password with, for example, this command:

shell> docker logs mysql1 2>&1 | grep GENERATED

GENERATED ROOT PASSWORD: Axegh3kAJyDLaRuBemecis&EShOs

#### Connecting to MySQL Server from within the Container

Once the server is ready, you can run the **mysql** client within the MySQL Server container you just started, and connect it to the MySQL Server. Use the **docker exec -it** command to start a **mysql** client inside the Docker container you have started, like the following:

docker exec -it mysql1 mysql -uroot -p

When asked, enter the generated root password (see the last step in [Starting a MySQL Server Instance](https://dev.mysql.com/doc/mysql-installation-excerpt/5.5/en/docker-mysql-getting-started.html#docker-starting-mysql-server) above on how to find the password). After you have connected a **mysql** client to the server, you should reset the server root password by issuing this statement:

mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'newpassword';

Substitute *newpassword* with the password of your choice. Once the password is reset, the server is ready for use.

#### Container Shell Access

To have shell access to your MySQL Server container, use the **docker exec -it** command to start a bash shell inside the container:

shell> docker exec -it mysql1 bash

bash-4.2#

You can then run Linux commands inside the container. For example, to view contents in the server's data directory inside the container, use this command:

bash-4.2# ls /var/lib/mysql

auto.cnf ca.pem client-key.pem ib\_logfile0 ibdata1 mysql mysql.sock.lock private\_key.pem server-cert.pem sys

ca-key.pem client-cert.pem ib\_buffer\_pool ib\_logfile1 ibtmp1 mysql.sock performance\_schema public\_key.pem server-key.pem

#### Stopping and Deleting a MySQL Container

To stop the MySQL Server container we have created, use this command:

docker stop mysql1

**docker stop** sends a SIGTERM signal to the **mysqld** process, so that the server is shut down gracefully.

Also notice that when the main process of a container (**mysqld** in the case of a MySQL Server container) is stopped, the Docker container stops automatically.

To start the MySQL Server container again:

docker start mysql1

To stop and start again the MySQL Server container with a single command:

docker restart mysql1

To delete the MySQL container, stop it first, and then use the **docker rm** command:

docker stop mysql1

docker rm mysql1