AN OPENSOURCE EBOOK

INTRODUCTION TO SQL

Bobby Iliev

- **This version was published on October 13,2021**

This open-source introduction to SQL guide will help you learn the basics of SQL and start using relational databases for your SysOps, DevOps, and Dev projects. Whether you are a DevOps/SysOps engineer, developer, or just a Linux enthusiast, you will most likely have to use SQL at some point in your career.

The guide is suitable for anyone working as a developer, system administrator, or DevOps engineer who wants to learn the basics of SQL.

My name is Bobby Iliev, and I have been working as a Linux DevOps Engineer since 2014. I am an avid Linux lover and supporter of the open-source movement philosophy. I am always doing that which I cannot do in order that I may learn how to do it, and I believe in sharing knowledge.

I think it's essential always to keep professional and surround yourself with good people, work hard, and be nice to everyone. You have to perform at a consistently higher level than others. That's the mark of a true professional.

For more information, please visit my blog at https://bobbyiliev.com, follow me on Twitter @bobbyiliev_ and YouTube.

This book is made possible thanks to these fantastic companies!

The Streaming Database for Real-time Analytics.

Materialize is a reactive database that delivers incremental view updates. Materialize helps developers easily build with streaming data using standard SQL.

DigitalOcean is a cloud services platform delivering the simplicity developers love and businesses trust to run production applications at scale.

It provides highly available, secure, and scalable compute, storage, and networking solutions that help developers build great software faster.

Founded in 2012 with offices in New York and Cambridge, MA, DigitalOcean offers transparent and affordable pricing, an elegant user interface, and one of the largest libraries of open source resources available.

For more information, please visit https://www.digitalocean.com or follow @digitalocean on Twitter.

If you are new to DigitalOcean, you can get a free $100 credit and spin up your own servers via this referral link here:

Free $100 Credit For DigitalOcean

The DevDojo is a resource to learn all things web development and web design. Learn on your lunch break or wake up and enjoy a cup of coffee with us to learn something new.

Join this developer community, and we can all learn together, build together, and grow together.

Join DevDojo

For more information, please visit https://www.devdojo.com or follow @thedevdojo on Twitter.

This ebook was generated by Ibis developed by Mohamed Said.

Ibis is a PHP tool that helps you write eBooks in markdown.

The cover for this ebook was created with Canva.com.

If you ever need to create a graphic, poster, invitation, logo, presentation – or anything that looks good — give Canva a go.

MIT License

Before we dive deep into SQL, let's quickly define what a database is.

The definition of databases from Wikipedia is:

A database is an organized collection of data, generally stored and accessed electronically from a computer system.

In other words, a database is a collection of data stored and structured in different database tables.

You've most likely worked with spreadsheet systems like Excel or Google Sheets. At the very basic, database tables are quite similar to spreadsheets.

Each table has different **columns** which could contain different types of data.

For example, if you have a todo list app, you would have a database, and in your database, you would have different tables storing different information like:

- Users - In the users table, you would have some data for your users like: `username`, `name`, and `active`, for example.
- Tasks - The tasks table would store all of the tasks that you are planning to do. The columns of the tasks table would be for example, `task_name`, `status`, `due_date` and `priority`.

The Users table will look like this:

```
+----+----------+---------------+--------+
| id | username | name          | active |
+----+----------+---------------+--------+
| 1  |    bobby | Bobby Iliev   |   true |
| 2  |    grisi | Greisi I.     |   true |
| 3  |  devdojo | Dev Dojo      |  false |
+----+----------+---------------+--------+
```

Rundown of the table structure:

- We have 4 columns: `id`, `username`, `name` and `active`.
- We also have 3 entries/users.
- The `id` column is a unique identifier of each user and is auto-incremented.

In the next chapter, we will learn how to install MySQL and create our first database.

Now that you know what a database, table, and column are, the next thing that you would need to do is install a database service where you would be running your SQL queries on.

We will be using MySQL as it is free, open-source, and very widely used.

Depending on your operating system, to install MySQL run the following commands.

To install MySQL on a Linux or Ubuntu machine, run the following commands:

- First update your `apt` repository:

```
sudo apt update -y
```

- Then install MySQL:

```
sudo apt install mysql-server mysql-client
```

We are installing two packages, one is the actual MySQL server, and the other is the MySQL client, which would allow us to connect to the MySQL server and run our queries.

To check if MySQL is running, run the following command:

```
sudo systemctl status mysql.service
```

To secure your MySQL server, you could run the following command:

```
sudo mysql_secure_installation
```

Then follow the prompt and choose a secure password and save it in a secure place like a password manager.

With that, you would have MySQL installed on your Ubuntu server. The above should also work just fine on Debian.

I would recommend installing MySQL using Homebrew:

```
brew install mysql
```

After that, start MySQL:

```
brew services start mysql
```

And finally, secure it:

```
mysql_secure_installation
```

In case that you ever need to stop the MySQL service, you could do so with the following command:

```
brew services stop mysql
```

To install MySQL on Windows, I would recommend following the steps from the official documentation here:

https://dev.mysql.com/doc/refman/8.0/en/windows-installation.html

To access MySQL run the `mysql` command followed by your user:

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

After that, switch to the `demo` database that we created in the previous chapter:

```
USE demo;
```

To exit the just type the following:

```
exit;
```

`.my.cnf`

By configuring the `~/.my.cnf` file in your user's home directory, MySQL would allow you to log in without prompting you for a password.

To make that change, what you need to do is first create a `.my.cnf` file in your user's home directory:

```
touch ~/.my.cnf
```

After that, set secure permissions so that other regular users could not read the file:

```
chmod 600 ~/.my.cnf
```

Then using your favourite text editor, open the file:

```
nano ~/.my.cnf
```

And add the following configuration:

```
[client]
user=YOUR_MYSQL_USERNAME
password=YOUR_MYSQL_PASSWORD
```

Make sure to update your MySQL credentials accordingly, then save the file and exit.

After that, if you run just `mysql`, you will be authenticated directly with the credentials that you've specified in the `~/.my.cnf` file without being prompted for a password.

As a quick test, you could check all of your open SQL connections by running the following command:

```
mysqladmin proc
```

The `mysqladmin` tool would also use the client details from the `~/.my.cnf` file, and it would list your current MySQL process list.

Another cool thing that you could try doing is combining this with the `watch` command and kind of monitor your MySQL connections in almost real-time:

```
watch -n1 mysqladmin proc
```

To stop the `watch` command, just hit `CTRL+C`

If you prefer using GUI clients, you could take a look a the following ones and install them locally on your laptop:

- MySQL Workbench
- Sequel Pro
- TablePlus

This will allow you to connect to your database via a graphical interface rather than the `mysql` command-line tool.

If you want to have a production-ready MySQL database, I would recommend giving DigitalOcean a try:

Worry-free managed database hosting

Before we get started with SQL, let's learn how to create tables and columns.

As an example, we are going to create a `users` table with the following columns:

- `id` - this is going to be the primary key of the table and would be the unique identifier of each user.
- `username` - this column would hold the username of our users.
- `name` - here, we will store the full name of users.
- `status` - here, we will store the status of a user, which would indicate if a user is active or not.

You need to specify the data type of each column.

In our case it would be like this:

- `id` - Integer
- `username` - Varchar
- `name` - Varchar
- `status` - Number

The most common data types that you would come across are:

- `CHAR`(size): Fixed-length character string with a maximum length of 255 bytes.
- `VARCHAR`(size): Variable-length character string. Max size is specified in parenthesis.
- `TEXT`(size): A string with a maximum length of 65,535 bytes.
- `INTEGER`(size) or `INT`(size): A medium integer.
- `BOOLEAN` or `BOOL`: Holds a true or false value.
- `DATE`: Holds a date.

Let's have the following users table as an example:

- `id`: We would want to set the ID to `INT`.
- `name`: The name should fit in a `VARCHAR` column.
- `about`: As the about section could be longer, we could set the column data type to `TEXT`.
- `birthday`: For the birthday column of the user, we could use `DATE`.

For more information on all data types available, make sure to check out the official documentation here.

As we briefly covered in the previous chapter, before you could create tables, you would need to create a database by running the following:

- First access MySQL:

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

- Then create a database called `demo_db`:

```
CREATE DATABASE demo_db;
```

Note: the database name needs to be unique, if you already have a database named `demo_db` you would receive an error that the database already exists.

You can consider this database as the container where we would create all of the tables in.

Once you've created the database, you need to switch to that database:

```
USE demo_db;
```

You can think of this as accessing a directory in Linux with the `cd` command. With `USE`, we switch to a specific database.

Alternatively, if you do not want to 'switch' to the specific database, you would need to specify the so-called fully qualified table name. For example, if you had a `users` table in the `demo_db`, and you wanted to select all of the entries from that table, you could use one of the following two approaches:

- Switch to the `demo_db` first and then run a select statement:

```
USE demo_db;
SELECT username FROM users;
```

- Alternatively, rather than using the `USE` command first, specify the database name followed by the table name separated with a dot: `db_name.table_name`:

```
SELECT username FROM demo_db.users;
```

We are going to cover the `SELECT` statement more in-depth in the following chapters.

In order to create a table, you need to use the `CREATE TABLE` statement followed by the columns that you want to have in that table and their data type.

Let's say that we wanted to create a `users` table with the following columns:

- `id`: An integer value
- `username`: A varchar value
- `about`: A text type
- `birthday`: Date
- `active`: True or false

The query that we would need to run to create that table would be:

```
CREATE TABLE users
(
    id INT,
    username VARCHAR(255),
    about TEXT,
    birthday DATE,
    active BOOL
);
```

Note: You need to select a database first with the `USE` command as mentioned above. Otherwise you will get the following error: `ERROR 1046 (3D000): No database selected.

To list the available tables, you could run the following command:

```
SHOW TABLES;
```

Output:

```
+--------------------+
| Tables_in_demo_db |
+--------------------+
| users              |
+--------------------+
```

You can create a new table from an existing table by using the `CREATE TABLE AS` statement.

Let's test that by creating a new table from the table `users` which we created earlier.

```
CREATE TABLE users2 AS
(
    SELECT * FROM users
);
```

The output that you would get would be:

```
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Note: When creating a table in this way, the new table will be populated with the records from the existing table (based on the SELECT Statement)