How To Manage an SQL Database

Posted September 26, 2018 17.4k MYSQL POSTGRESQL MARIADB DATABASES

By: Mark Drake

An SQL Cheat Sheet

Introduction

SQL databases come installed with all the commands you need to add, modify, delete, and query your data. This cheat sheet-style guide provides a quick reference to some of the most commonly-used SQL commands.

How to Use This Guide:

- This guide is in cheat sheet format with self-contained command-line snippets
- Jump to any section that is relevant to the task you are trying to complete

- When you see `highlighted text` in this guide's commands, keep in mind that this text should refer to the columns, tables, and data in *your own* database.
- Throughout this guide, the example data values given are all wrapped in apostrophes ('). In SQL, it is necessary to wrap any data values that consist of strings in apostrophes. This isn't required for numeric data, but it also won't cause any issues if you do include apostrophes.

Please note that, while SQL is recognized as a standard, most SQL database programs have their own proprietary extensions. This guide uses MySQL as the example relational database management system (RDBMS), but the commands given will work with other relational database programs, including PostgreSQL, MariaDB, and SQLite. Where there are significant differences between RDBMSs, we have included the alternative commands.

Opening up the Database Prompt (using Socket/Trust Authentication)

By default on Ubuntu 18.04, the **root** MySQL user can authenticate without a password using the following command:

```
$ sudo mysql
```

To open up a PostgreSQL prompt, use the following command. This example will log you in as the **postgres** user, which is the included superuser role, but you can replace that with any already-created role:

```
$ sudo -u postgres psql
```

Opening up the Database Prompt (using Password Authentication)

If your **root** MySQL user is set to authenticate with a password, you can do so with the following command:

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

If you've already set up a non-root user account for your database, you can also use this method to log in as that user:

```
$ mysql -u user -p
```

The above command will prompt you for your password after you run it. If you'd like to supply your password as part of the command, immediately follow the `-p` option with your password, with no space between them:

```
$ mysql -u root -ppassword
```

Creating a Database

The following command creates a database with default settings.

```
msyql> CREATE DATABASE database_name;
```

If you want your database to use a character set and collation different than the defaults, you can specify those using this syntax:

```
mysql> CREATE DATABASE database_name CHARACTER SET character_set COLLATE collation;
```

Listing Databases

To see what databases exist in your MySQL or MariaDB installation, run the following command:

```
mysql> SHOW DATABASES;
```

In PostgreSQL, you can see what databases have been created with the following command:

```
postgres=# \list
```

Deleting a Database

To delete a database, including any tables and data held within it, run a command that follows this structure:

```
mysql> DROP DATABASE IF EXISTS database;
```

Creating a User

To create a user profile for your database without specifying any privileges for it, run the following command:

```
mysql> CREATE USER username IDENTIFIED BY 'password';
```

PostgreSQL uses a similar, but slightly different, syntax:

```
postgres=# CREATE USER user WITH PASSWORD 'password';
```

If you want to create a new user and grant them privileges in one command, you can do so by issuing a `GRANT` statement. The following command creates a new user and grants them full privileges to every database and table in the RDBMS:

```
mysql> GRANT ALL PRIVILEGES ON *.* TO 'username'@'localhost' IDENTIFIED BY 'password';
```

Note the `PRIVILEGES` keyword in this previous `GRANT` statement. In most RDBMSs, this keyword is optional, and this statement can be equivalently written as:

```
mysql> GRANT ALL ON *.* TO 'username'@'localhost' IDENTIFIED BY 'password';
```

Be aware, though, that the `PRIVILEGES` keyword is required for granting privileges like this when Strict SQL mode is turned on.

Deleting a User

Use the following syntax to delete a database user profile:

```
mysql> DROP USER IF EXISTS username;
```

Note that this command will not by default delete any tables created by the deleted user, and attempts to access such tables may result in errors.

Selecting a Database

Before you can create a table, you first have to tell the RDBMS the database in which you'd like to create it. In MySQL and MariaDB, do so with the following syntax:

```
mysql> USE database;
```

In PostgreSQL, you must use the following command to select your desired database:

```
postgres=# \connect database
```

Creating a Table

The following command structure creates a new table with the name `table`, and includes two columns, each with their own specific data type:

```
mysql> CREATE TABLE table ( column_1 column_1_data_type, column_2 column_2_data_taype );
```

Deleting a Table

To delete a table entirely, including all its data, run the following:

```
mysql> DROP TABLE IF EXISTS table
```

Inserting Data into a Table

Use the following syntax to populate a table with one row of data:

```
mysql> INSERT INTO table ( column_A, column_B, column_C ) VALUES ( 'data_A', 'data_B', 'data_C' );
```

You can also populate a table with multiple rows of data using a single command, like this:

```
mysql> INSERT INTO table ( column_A, column_B, column_C ) VALUES ( 'data_1A', 'data_1B', 'data_1C' )
```

Deleting Data from a Table

To delete a row of data from a table, use the following command structure. Note that `value` should be the value held in the specified `column` in the row that you want to delete:

```
mysql> DELETE FROM table WHERE column='value';
```

Note: If you do not include a `WHERE` clause in a `DELETE` statement, as in the following example, it will delete all the data held in a table, but not the columns or the table itself:

```
mysql> DELETE FROM table;
```

Changing Data in a Table

Use the following syntax to update the data held in a given row. Note that the `WHERE` clause at the end of the command tells SQL which row to update. `value` is the value held in `column_A` that aligns with the row you want to change.

> **Note:** If you fail to include a `WHERE` clause in an `UPDATE` statement, the command will replace the data held in every row of the table.

```
mysql> UPDATE table SET column_1 = value_1, column_2 = value_2 WHERE column_A=value;
```

Inserting a Column

The following command syntax will add a new column to a table:

```
mysql> ALTER TABLE table ADD COLUMN column data_type;
```

Deleting a Column

A command following this structure will delete a column from a table:

```
mysql> ALTER TABLE table DROP COLUMN column;
```

Performing Basic Queries

To view all the data from a single column in a table, use the following syntax:

```
mysql> SELECT column FROM table;
```

To query multiple columns from the same table, separate the column names with a comma:

```
mysql> SELECT column_1, column_2 FROM table;
```

You can also query every column in a table by replacing the names of the columns with an asterisk (*). In SQL, asterisks act as placeholders to represent “all”:

```
mysql> SELECT * FROM table;
```

Using WHERE Clauses

You can narrow down the results of a query by appending the `SELECT` statement with a `WHERE` clause, like this:

```
mysql> SELECT column FROM table WHERE conditions_that_apply;
```

For example, you can query all the data from a single row with a syntax like the following. Note that `value` should be a value held in both the specified `column` and the row you want to query:

```
mysql> SELECT * FROM table WHERE column = value;
```

Working with Comparison Operators

A comparison operator in a `WHERE` clause defines how the specified column should be compared against the value. Here are some common SQL comparison operators:

Operator	What it does
`=`	tests for equality
`!=`	tests for inequality
`<`	tests for less-than
`>`	tests for greater-than
`<=`	tests for less-than or equal-to
`>=`	tests for greater-than or equal-to
`BETWEEN`	tests whether a value lies within a given range
`IN`	tests whether a row's value is contained in a set of specified values
`EXISTS`	tests whether rows exist, given the specified conditions
`LIKE`	tests whether a value matches a specified string
`IS NULL`	tests for `NULL` values
`IS NOT NULL`	tests for all values other than `NULL`

Working with Wildcards

SQL allows the use of wildcard characters. These are useful if you're trying to find a specific entry in a table, but aren't sure of what that entry is exactly.

Asterisks (`*`) are placeholders that represent “all,” this will query every column in a table:

```
mysql> SELECT * FROM table;
```

Percentage signs (`%`) represent zero or more unknown characters.

```
mysql> SELECT * FROM table WHERE column LIKE val%;
```

Underscores (`_`) are used to represent a single unknown character:

```
mysql> SELECT * FROM table WHERE column LIKE v_lue;
```

Counting Entries in a Column

The `COUNT` function is used to find the number of entries in a given column. The following syntax will return the total number of values held in `column`:

```
mysql> SELECT COUNT(column) FROM table;
```

You can narrow down the results of a `COUNT` function by appending a `WHERE` clause, like this:

```
mysql> SELECT COUNT(column) FROM table WHERE column=value;
```

Finding the Average Value in a Column

The `AVG` function is used to find the average (in this case, the mean) amongst values held in a specific column. Note that the `AVG` function will only work with columns holding numeric values; when used on a column holding string values, it may return an error or `0`:

```
mysql> SELECT AVG(column) FROM table;
```

Finding the Sum of Values in a Column

The `SUM` function is used to find the sum total of all the numeric values held in a column:

```
mysql> SELECT SUM(column) FROM table;
```

As with the `AVG` function, if you run the `SUM` function on a column holding string values it may return an error or just `0`, depending on your RDBMS.

Finding the Largest Value in a Column

To find the largest numeric value in a column or the last value alphabetically, use the `MAX` function:

```
mysql> SELECT MAX(column) FROM table;
```

Finding the Smallest Value in a Column

To find the smallest numeric value in a column or the first value alphabetically, use the `MIN` function:

```
mysql> SELECT MIN(column) FROM table;
```

Sorting Results with ORDER BY Clauses

An `ORDER BY` clause is used to sort query results. The following query syntax returns the values from `column_1` and `column_2` and sorts the results by the values held in `column_1` in ascending order or, for string values, in alphabetical order:

```
mysql> SELECT column_1, column_2 FROM table ORDER BY column_1;
```

To perform the same action, but order the results in descending or reverse alphabetical order, append the query with `DESC`:

```
mysql> SELECT column_1, column_2 FROM table ORDER BY column_1 DESC;
```

Sorting Results with GROUP BY Clauses

The `GROUP BY` clause is similar to the `ORDER BY` clause, but it is used to sort the results of a query that includes an aggregate function such as `COUNT`, `MAX`, `MIN`, or `SUM`. On their own, the aggregate functions described in the previous section will only return a single value. However, you can view the results of an aggregate function performed on every matching value in a column by including a `GROUP BY` clause.

The following syntax will count the number of matching values in `column_2` and group them in ascending or alphabetical order:

```
mysql> SELECT COUNT(column_1), column_2 FROM table GROUP BY column_2;
```

To perform the same action, but group the results in descending or reverse alphabetical order, append the query with `DESC`:

```
mysql> SELECT COUNT(column_1), column_2 FROM table GROUP BY column_2 DESC;
```

Querying Multiple Tables with JOIN Clauses

`JOIN` clauses are used to create result-sets that combine rows from two or more tables. A `JOIN` clause will only work if the two tables each have a column with an identical name and data type, as in this example:

```
mysql> SELECT table_1.column_1, table_2.column_2 FROM table_1 JOIN table_2 ON table_1.common_column=
```

This is an example of an `INNER JOIN` clause. An `INNER JOIN` will return all the records that have matching values in both tables, but won't show any records that don't have matching values.

It's possible to return all the records from one of two tables, including values that do not have a corresponding match in the other table, by using an *outer* `JOIN` clause. Outer `JOIN` clauses are written as either `LEFT JOIN` or `RIGHT JOIN`.

A `LEFT JOIN` clause returns all the records from the "left" table and only the matching records from the "right" table. In the context of outer `JOIN` clauses, the left table is the one referenced in the `FROM` clause, and the right table is any other table referenced after the `JOIN` statement. The following will show every record from `table_1` and only the matching values from `table_2`. Any values that do not have a match in `table_2` will appear as `NULL` in the result-set:

```
mysql> SELECT table_1.column_1, table_2.column_2 FROM table_1 LEFT JOIN table_2 ON table_1.common_co
```

A `RIGHT JOIN` clause functions the same as a `LEFT JOIN`, but it prints the all the results from the right table, and only the matching values from the left:

```
mysql> SELECT table_1.column_1, table_2.column_2 FROM table_1 RIGHT JOIN table_2 ON table_1.common_c
```

Combining Multiple SELECT Statements with UNION Clauses

A `UNION` operator is useful for combining the results of two (or more) `SELECT` statements into a single result-set:

```
mysql> SELECT column_1 FROM table UNION SELECT column_2 FROM table;
```

Additionally, the `UNION` clause can combine two (or more) `SELECT` statements querying different tables into the same result-set:

```
mysql> SELECT column FROM table_1 UNION SELECT column FROM table_2;
```

Conclusion

This guide covers some of the more common commands in SQL used to manage databases, users, and tables, and query the contents held in those tables. There are, however, many combinations of clauses and operators that all produce unique result-sets. If you're looking for a more comprehensive guide to working with SQL, we encourage you to check out Oracle's Database SQL Reference.

Additionally, if there are common SQL commands you'd like to see in this guide, please ask or make suggestions in the comments below.

Related Tutorials

How To Ensure Code Quality with SonarQube on Ubuntu 18.04

How To Sync and Share Your Files with Seafile on Ubuntu 18.04

How To Troubleshoot Issues in MySQL

How To Set Up a Remote Database to Optimize Site Performance with MySQL on Ubuntu 18.04

How To Set Up Laravel, Nginx, and MySQL with Docker Compose

0 Comments

Leave a comment...

Log In to Comment