**The first thing you need to do is start the terminal.** Do that by clicking the "hamburger" menu at the top left of the screen, going to the "terminal" section, and clicking "new terminal". Once you open a new one, type echo hello PostgreSQL into the terminal and press enter.

Your virtual machine comes with PostgreSQL installed. You will use the Psql terminal application to interact with it. Log in by typing psql --username=freecodecamp --dbname=postgres into the terminal and pressing enter.

Notice that the prompt changed to let you know that you are now interacting with PostgreSQL. First thing to do is see what databases are here. Type \l into the prompt to **l**ist them.

The databases you see are there by default. You can make your own like this:

CREATE DATABASE database\_name;

The capitalized words are keywords telling PostgreSQL what to do. The name of the database is the lowercase word. Note that **all commands need a semi-colon at the end.** Create a new database named first\_database.

Use the **l**ist shortcut command again to make sure your new database is there.

It worked. Your new database is there. If you don't get a message after entering a command, it means it's incomplete and you likely forgot the semi-colon. You can just add it on the next line and press enter to finish the command. Create another database named second\_database.

You should have another new database now. **L**ist the databases to make sure.

You can **c**onnect to a database by entering \c database\_name. You need to connect to add information. Connect to your second\_database.

You should see a message that you are connected. Notice that the prompt changed to second\_database=>. So the postgres=> prompt before must have meant you were connected to that database. A database is made of tables that hold your data. Enter \d to **d**isplay the tables.

Looks like there's no tables or relations yet. Similar to how you created a database, you can create a table like this:

CREATE TABLE table\_name();

Note that the parenthesis are needed for this one. It will create the table in the database you are connected to. Create a table named first\_table in second\_database.

View the tables in second\_database again with the **d**isplay command. You should see your new table there with a little meta data about it.

Create another new table in this database. Give it a name of second\_table.

There should be two tables in this database now. **D**isplay them again to make sure.

You can view more details about a table by adding the table name after the **d**isplay command like this: \d table\_name. View more details about your second\_table.

Tables need **columns** to describe the data in them, yours doesn't have any yet. Here's an example of how to add one:

ALTER TABLE table\_name ADD COLUMN column\_name DATATYPE;

Add a column to second\_table named first\_column. Give it a data type of INT. INT stands for integer. Don't forget the semi-colon. 😄

Looks like it worked. **D**isplay the details of second\_table again to see if your new column is there.

Your table should have an id column added. View the details of second\_table to make sure.

Add another column to second\_table named age. Give it a data type of INT.

Those are some good looking columns. You will probably need to know how to remove them. Here's an example:

ALTER TABLE table\_name DROP COLUMN column\_name;

Drop your age column.

View the details of second\_table to see if it's gone.

It's gone. Use the ALTER TABLE and DROP COLUMN keywords again to drop first\_column.

A common data type is VARCHAR. It's a short string of characters. You need to give it a maximum length when using it like this: VARCHAR(30).

Add a new column to second\_table, give it a name of name and a data type of VARCHAR(30).

Take a look at the details of second\_table to see your columns.

You can see the VARCHAR type there. The 30 means the data in it can be a max of 30 characters. You named that column name, it should have been username. Here's how you can rename a column:

ALTER TABLE table\_name RENAME COLUMN column\_name TO new\_name;

Rename the name column to username.

Take a look at the details of second\_table again to see if it got renamed.

It worked. Rows are the actual data in the table. You can add one like this:

INSERT INTO table\_name(column\_1, column\_2) VALUES(value1, value2);

Insert a row into second\_table. Give it an id of 1, and a username of Samus. The username column expects a VARCHAR, so you need to put Samus in single quotes like this: 'Samus'.

You should have one row in your table. You can view the data in a table by querying it with the SELECT statement. Here's how it looks:

SELECT columns FROM table\_name;

Use a SELECT statement to view **all** the columns in second\_table. Use an asterisk (\*) to denote that you want to see all the columns.

There's your one row. **Insert** another row **into** second\_table. Fill in the id and username columns with the **values** 2 and 'Mario'.

You should now have two rows in the table. Use SELECT again to view **all** the columns and rows **from** second\_table.

**Insert** another row **into** second\_table. Use 3 as the id, and Luigi as the username this time.

You should now have three rows. Use SELECT again to see **all** the data you entered.

That gives me an idea 😃 You can make a database of Mario video game characters. You should start from scratch for it. Why don't you delete the record you just entered. Here's an example of how to delete a row:

DELETE FROM table\_name WHERE condition;

Remove Luigi from your table. The condition you want to use is username='Luigi'.

Luigi should be gone. Use SELECT again to see all the data and make sure he's not there.

It's gone. You can scrap all this for the new database. **Delete** Mario **from** second\_table using the same command as before, except make the condition username='Mario' this time.

Only one more row should remain. **Delete** Samus **from** second\_table.

Use SELECT again to see all the rows in second\_table to make sure they're gone.

Looks like they're all gone. Remind yourself what columns you have in second\_table by looking at its **d**etails.

There's two columns. You won't need either of them for the Mario database. **Alter** the **table** second\_table and **drop** the **column** username.

Next, drop the id column.

Okay, the table has no rows or columns left. View the tables in this database to see what is here.

Still two. You won't need either of those for the new database either. Drop second\_table from your database. Here's an example:

DROP TABLE table\_name;

All the tables are gone now, too. View all the databases using the command to **l**ist them.

Rename first\_database to mario\_database. You can rename a database like this:

ALTER DATABASE database\_name RENAME TO new\_database\_name;

List the databases to make sure it got renamed.

**C**onnect to your newly named database so you can start adding your characters.

List the databases again to make sure it's gone.

Okay, I think you're ready to get started. I don't think you created any tables here, take a look to make sure.

1. Use the **d**isplay shortcut command

Get A Hint

Create a new table named characters, it will hold some basic information about Mario characters.

Next, you can add some columns to the table. Add a column named character\_id to your new table that is a type of SERIAL.

1. Use the ALTER TABLE and ADD COLUMN keywords

The SERIAL type will make your column an INT with a NOT NULL constraint, and automatically increment the integer when a new row is added. View the details of the characters table to see what SERIAL did for you.

mario\_database=> \d characters

mario\_database=> Table "public.characters"

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

| Column | Type | Collation | Nullable | Default |

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

| character\_id | integer | | not null | nextval('characters\_character\_id\_seq'::regclass) |

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