Columns:

ID (integer identifier, unique to each row)

Name (text)

Address line 1(text)

Address line 2 (text)

City (integer identifier, drawn from separate table of cities, from which any stat or country info would be drawn)

Postal Code (text)

Industry (integer identifier, drawn form a separate table of industries)

Etc.

|  |  |  |
| --- | --- | --- |
|  | **Column 1** | **Column 2** |
| **Row 1** | Row 1, Column 1 | Row 1, Column 2 |
| **Row 2** | Row 2, Column 1 | Row 2, Column 2 |
| **Row 3** | Row 3, Column 1 | Row 3, Column 2 |

MY SQL

**SQL**

For any relational database like MySQL, you will interact with it using SQL. In the previous chapter, we learned how to design a schema for our data: we set up the collections that we needed, and we set up the relationships among tables. Now we will see the importance of relationships, and how to use SQL to adjust the data in any way we can imagine.

SQL stands for**Structured Query Language**, which is a programming language designed for managing data in relational databases. SQL statements are used to perform tasks; they can **SELECT** data, **SELECT** data **WHERE** some conditions are true, **INSERT** data, **UPDATE** data, **DELETE**data, and **JOIN** different tables together. As we go over all of the basic SQL commands, be patient. You will be learning a domain-specific language, unrelated to languages you may have previously seen. However, mastering SQL is the key to mastering the database component of your application.

**Database and SQL**

We will use MAMP (or WAMP) along with MySQL Workbench to interact with our database. MAMP plays an important role by setting up two different servers -- the **web server** (called Apache) and the **database server** (running MySQL Server). These servers each run on their own port. *For this chapter, we are interested in the database server.* We first must figure out which port the DB server is listening to, and then we need to configure MySQL Workbench to connect to that port. Then, we can use MySQL workbench to send SQL commands to the DB server. We will walk you through the initial connection in the next tab.

# Connecting to MySQL Server

Here are simple steps on how we can get started with MySQL:

1. Open MAMP, specify which port you want the Database Server to listen to, then start the server.
2. Open up MySQL Workbench, create a new connection to the port we specified in the previous step.

Note: for Windows users, newer versions of MySQL may require 'root' as the password.

# Importing Data

There are two main ways to import data into your MySQL Workbench. If you have an SQL file, you can just copy and paste the commands into the editor and click run. If you have an ERD diagram, you can forward engineer into MySQL workbench. We will go over how to do both ways.

From MySQL workbench, when inside the ERD. Top menu of Database/Forward Engineer

Make sure the Port is correct. Click on continue.continue/ pw: root. Continue

Continue or copy sql commands.

Continue.. go to connection from the MySQL.

# SELECT Basics

First, let's go over how we can **SELECT**data from a database. First, import the [twitter.sql](http://s3.amazonaws.com/General_V88/boomyeah/company_209/chapter_3569/handouts/chapter3569_5436_twitter.sql) into your MySQL workbench. The twitter.sql file contains the SQL statements to create a database called**twitter**along with certain tables and pre-populated fields.

Remember that you are learning a new language. Watch the videos once, then watch it a second time following along. Also, make sure you run all of the commands listed out in this tab. Even though you might understand it conceptually, it is important that we type the commands so we can retain our knowledge.

If you have a sql file, you can open it and copy the contents and paste it into your MySQL. Press lightning bolt. Refresh from the left side column.

Double click the new database.. lower left column

## Basics

What query would you run to get all of the users?

SELECT \* FROM users;

What query would you run to only get the first names of all of the users?

SELECT first\_name FROM users;

What query would you run to only get the first and last names of all of the users?

SELECT first\_name, last\_name FROM users;

## SELECT w/ Conditionals

What query would you run to only get the first name of users with id of 2?

SELECT first\_name FROM users WHERE id = 2;

What query would you run to get the last names of users with id of 2 or 3?

SELECT last\_name FROM users WHERE id = 2 OR id = 3;

What query would you run to get all of the users with id greater than 2?

SELECT \* FROM users WHERE id > 2;

What query would you run to get all of the users with id less than or equal to 3?

SELECT \* FROM users WHERE id <= 3;

What query would you run to get all of the users with first names ending in "e"?

SELECT \* FROM users WHERE first\_name LIKE "%e";

What query would you run to get all of the users with first names starting in "K"?

SELECT \* FROM users WHERE first\_name LIKE "K%";

## SELECT w/ Sorting

What query would you run to get all of the users with the youngest users at the top of the table?

SELECT \* FROM users ORDER BY birthday DESC;

What query would you run to get all of the users with the oldest users at the top of the table?

SELECT \* FROM users ORDER BY birthday ASC;

What query would you run to get all of the users with the first name that ends with "e" with the youngest users at the top of the table?

SELECT \* FROM users WHERE first\_name LIKE "%e" ORDER BY birthday DESC;

What query would you run to get only the first names of all of the users in alphabetical order?

SELECT first\_name FROM users ORDER BY first\_name;

The default for ORDER BY is ASC so we can leave that part out if we want the sorting to be ascending.

### Note

**Before moving on to the next tab, it will be best to go over the following tutorials on SQL Zoo:**

* SELECT Basics: <http://sqlzoo.net/wiki/SQLZOO:SELECT_basics>
* SELECT name: <http://sqlzoo.net/wiki/SELECT_names>
* SELECT from World: <http://sqlzoo.net/wiki/SELECT_from_WORLD_Tutorial>

# INSERT Basics

Now, let's go over how we can INSERT data. We can do this in two ways. One way is to manipulate the table directly using our MySQL Workbench and another way is to run direct SQL commands in the editor.

Even if you are using the GUI to insert data, it is important to see what kind of SQL commands it is running. We are going to have to run INSERT statements using SQL commands later so it is important that we know how to INSERT data both ways.

## Inserting Records

The SQL command pattern for INSERTing records is as follows:

INSERT INTO table\_name (column\_name1, column\_name2) VALUES('column1\_value', 'column2\_value');

# UPDATE Basics

We can UPDATE our database in two ways as well. Once again, it is important to pay attention to the SQL commands that are being run even if we just use the GUI because we will have to run UPDATE commands later in the bootcamp. Try updating your table both ways.

## Updating Records

The SQL command pattern for updating/editing records is as follows:

UPDATE table\_name SET column\_name1 = 'some\_value', column\_name2='another\_value' WHERE condition(s)

**IMPORTANT**: if **WHERE** condition is not added to the **UPDATE** statement, the changes will be applied to every record in the table.

# DELETE Basics

You can DELETE your records as well.

If you are getting an error regarding SQL SAFE UPDATES, run the following command to let MySQL Workbench know that you know what you are doing and you want to DELETE stuff from the database.

SET SQL\_SAFE\_UPDATES = 0;

## Deleting Records

The SQL command pattern for deleting/removing records is as follows:

DELETE FROM table\_name WHERE condition(s)

**IMPORTANT**: if **WHERE** condition is not added to the **DELETE** statement, it will delete all the records on the table.

# Fun with Functions

Go ahead and follow along with the video to practice more SQL commands.

Functions can be applied to the selected columns. There can be a variety of reasons why you might want to use functions. Below are some of the most commonly used functions separated by the purpose for using them. Get familiar with how to use some of these functions, but DO NOT TRY TO MEMORIZE ALL OF THESE. If you understand how functions work, then you should be able to quickly reference a function and understand how to use it.

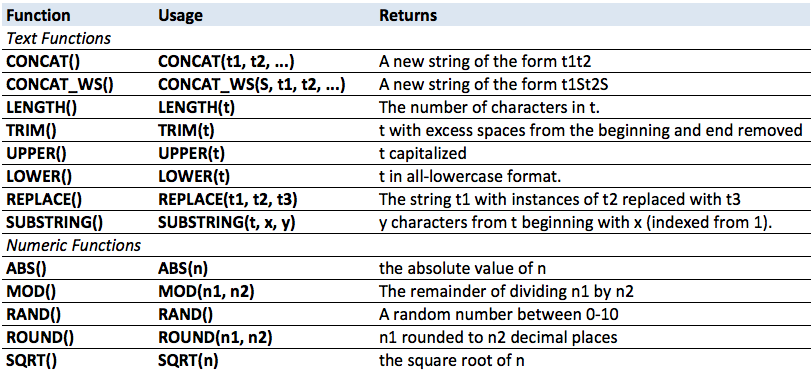
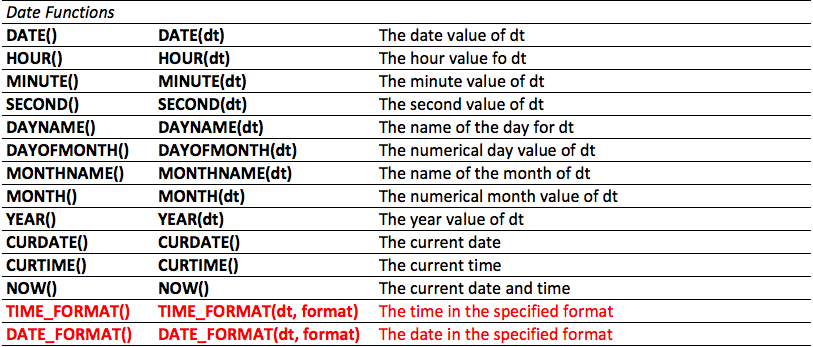
When calling a function on a column, make sure that column is the appropriate Data Type for that function.

**Text Functions Data Types (**VARCHAR, TEXT, CHAR etc.**)**

**Numeric Functions Data Types (**INT, BIGINT, FLOAT etc.**)**

**Date and Time Functions Data Types (**DATETIME**)**

**SELECT** ***FUNC­­­TION (****column****)*** **FROM** *table\_name*

To see the different types of formatting for TIME\_FORMAT() and DATE\_FORMAT() look here:  [SQL Date Format](http://www.w3schools.com/sql/func_date_format.asp" \t "_blank).

Again, DO NOT TRY TO MEMORIZE THESE FUNCTIONS. You should be aware of the types of functions that exist, and look up the details only when needed. For example, the RAND() function above *actually*returns a decimal number as low as **0** and as high as "**almost 1.0**". Is this detail worth memorizing? No! Just become familiar with them, and develop a keen eye for reading documentation and quickly understanding functions.

[www.w3schools.com/sql/func\_date\_format.asp](http://www.w3schools.com/sql/func_date_format.asp)

for more functions ***syntax***

# Joining Tables

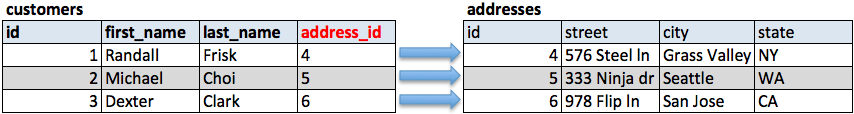
Remember *foreign keys* from the last chapter? We now get to put them to use! We JOIN two tables on the*ids*that match each other. This means that we can't JOIN tables together that don't have a relationship with each other (e.g. **One to One**, **One to Many**, **Many to Many**). A*foreign key* in a table matches up directly with an id in another table. Let's take a peek at what this looks like.

Go ahead and download the  [morepractice.sql](http://s3.amazonaws.com/General_V88/boomyeah/company_209/chapter_3569/handouts/chapter3569_5437_morepractice.sql) and follow along with the video.

Here is another example:

### ****One to One****

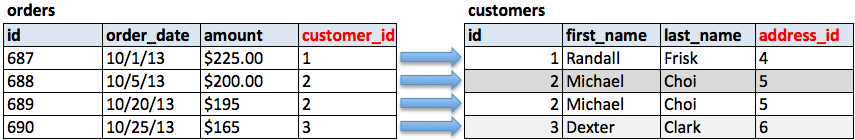
SELECT \* FROM customers JOIN addresses ON addresses.id = customers.address\_id;

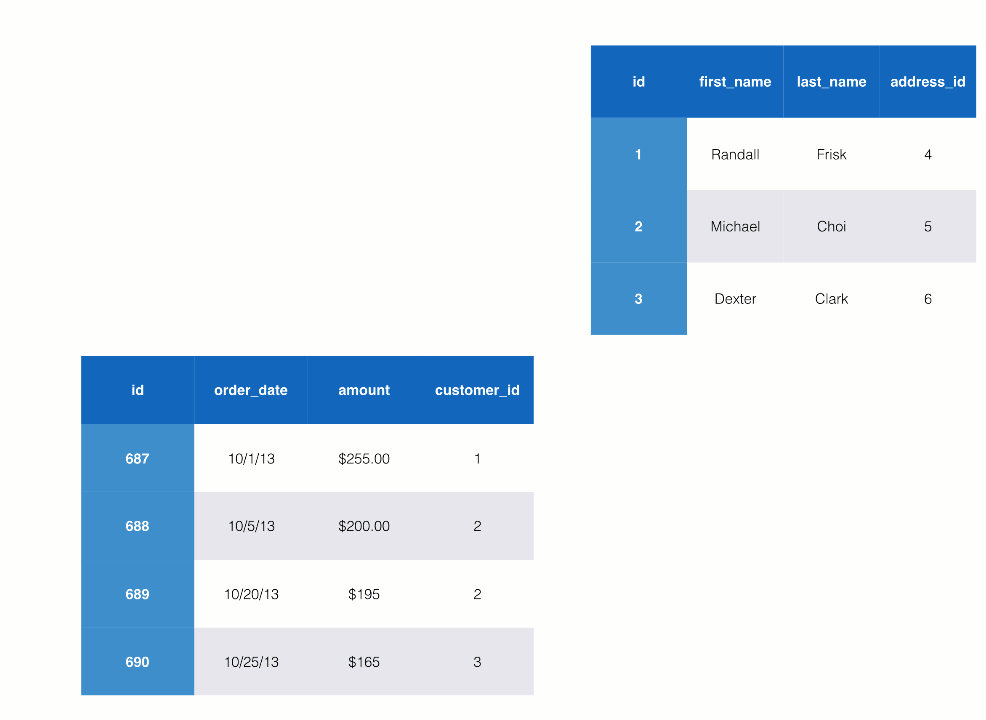


## http://i.imgur.com/0Bw6pb7.gif

### ****One to Many****

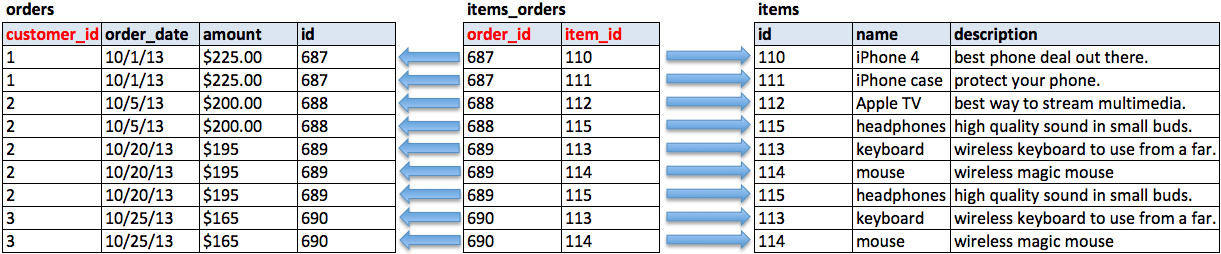
SELECT \* FROM orders JOIN customers ON customers.id = orders.customer\_id;

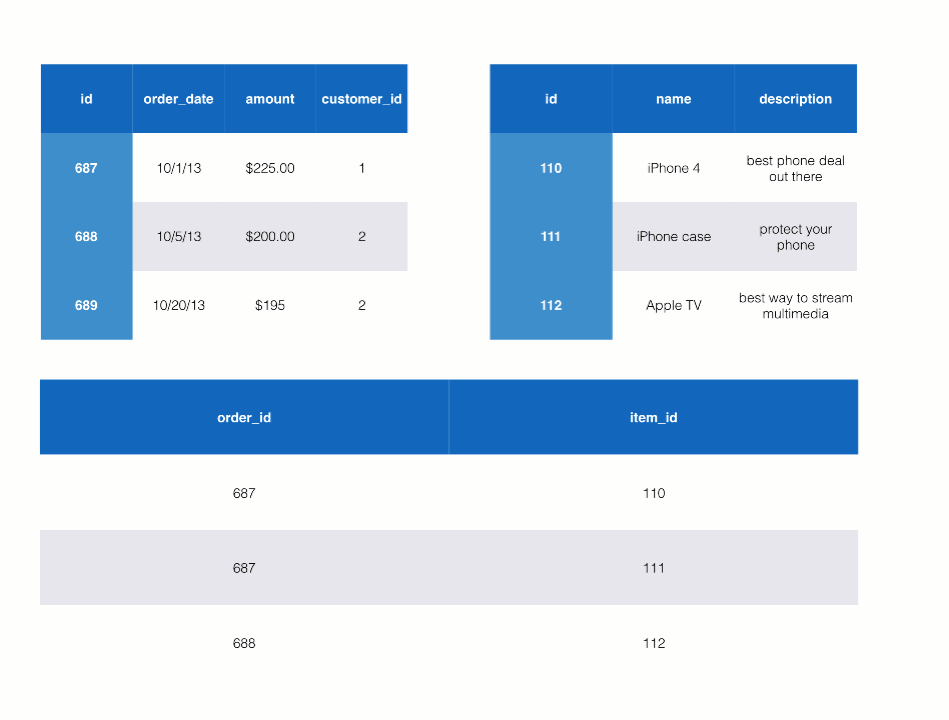


****

### ****Many to Many****

SELECT \* FROM orders JOIN items\_orders ON orders.id = items\_orders.order\_id JOIN items ON items.id = items\_orders.item\_id;





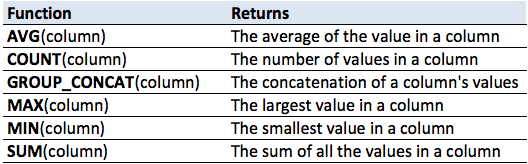
The above examples are using a JOIN, also referred to as INNER JOIN. However, there may be times where you would want to use a different type of JOIN. Imagine that in the  **One to Many** relationship above there was a customer that had not yet placed an order. There wouldn't be a customer\_id listed in the **orders** table. If we wanted to get the results of all the customers orders including the customers that hadn't yet placed an order we would have to use a **LEFT JOIN**. See the visual representations of what you can expect to be included when using different types of joins.

## Grouping Results

In the previous section, we saw how we could use functions to manipulate a single value in a single row. With **GROUP BY**, we will group multiple rows together, by performing a function to combine the values of those rows. Because this results in a single result for the group, it will combine those grouped rows into a single resultant row.

As you can imagine, there are many different ways that we might combine multiple values into a result. Below are a list of the most common ones, often called Grouping Functions or Aggregate Functions.

**Aggregate Functions**



# Left Join

In addition to the basic JOIN, there are many other types of joins that you can do in SQL. However, you can make any web application using only JOIN and LEFT JOIN. This is why we advise our students that once they understand the basic JOIN, to focus only on learning the LEFT JOIN. These two are enough.

Only after you have mastered LEFT JOIN should you move on to additional JOINs. It is very important for you to run all of the following commands in this tab and visualize what is happening. We are joining the tables, starting from left to right, gluing each table based on the primary id and foreign key. This is why we set up relationships so we can do LEFT JOINs and create customized tables when we need them.

First, study the ERD below. Then run the commands that follow.