**Lesson 1 Basic SQL:**

* SQL: language that allows us to access data stored in a database
* Database: collection of tables that share connected data stored in a computer
* Entity Relationship Diagrams (ERD): Diagram that shows how data is structured in a database

Why use SQL?

* SQL is easy to understand (NOT case sensitive!)
* Can be used to directly access large amounts of data
* Easy to audit and replicate
* Can run multiple queries across multiple tables at once

How do databases store data?

* Data is stored in tables similar to excel spreadsheets
* All data in same column must match in terms of data type(i.e. text, numerical)
  + Enables database to pull data quickly
* Statements: a piece of correctly written data that tells the database what you’d like to do with data
  + **CREATE:** creates new table in database

These statements CHANGE the data in the database.

* + **DROP\_TABLE:** removes table in database
  + **SELECT:** allows you to read data and display it (called **query**)
    - **FROM:** specifies from which table(s) you want to use in the query

SELECT \* FROM

SELECT id, occurred\_at

* + - * can also select all tables with an \*
      * separate tables you want to pull up with a ,
    - SQL is not case sensitive, HOWEVER it is best practice to keep all SQL commands in caps and everything else in query lower case
    - Avoid spaces in column names! You can add as many spaces and blank likes between code as you want however
    - It is considered best practice to put a semicolon at the end of each statement, which also allows you to run as many queries at once.
  + **LIMIT:** useful when you just want to see the first few rows of a table. ALWAYS the last part of a query!
  + **ORDER BY:** allows you to sort results using the data in any column (i.e. if you need to sort orders by date, etc.)
    - MUST write this statement between FROM and LIMIT statements, or query will not run!
    - DESC can be added after ORDER BY statement to sort results into descending order as default is to sort in ascending order
    - Can also use for more than one column at a time
      * Sorting will happen in order in which you specify the columns
  + **WHERE:** Can display subsets of tables based on conditions that must be met
    - Filtering the data
    - Goes after FROM but before ORDER BY statements
    - Clauses MUST be in correct order or query will return an error
    - Can also use comparison operators with non-numeric data ( = and != only)
      * MUST put values that are non-numeric in single quotes!!
      * >
      * <
      * >=
      * <=
      * !=

Derived Columns: A new column that is a manipulation of the existing columns in your database

* Can include simple arithmetic or any advanced calculation
* Generally only temporary, existing only for the duration of your query
* Remember PEMDAS!!

Logical Operators:

* **LIKE** This allows you to perform operations similar to using **WHERE** and =, but for cases when you might **not** know **exactly** what you are looking for.
  + **Pulls all column values which are similar to specified characters (when you don’t know exactly what you’re looking for)**
  + Useful when working with text. Will use within WHERE clause
  + Frequently used with %, which tells us that we might want any number of characters leading up to a particular set of characters or following a certain set of characters.
  + LIKE operator cannot deal with date values, only text values
* **IN** This allows you to perform operations similar to using **WHERE** and =, but for more than one condition.
  + **Pulls column values which are strictly equal to any value from a specified set (when you know exactly what you’re looking for)**
  + Useful for both text and numeric columns
  + Requires single quotation marks around non-numerical data, numerical data can be entered directly
* **NOT** This is used with **IN** and **LIKE** to select all of the rows **NOT LIKE** or **NOT IN** a certain condition.
  + Useful for working with LIKE and IN
  + Can grab all rows that do not meet particular criteria
  + Type in NOT before IN or LIKE in code
* **AND & BETWEEN** These allow you to combine operations where all combined conditions must be true.
  + AND allows you to run two complete logical statements. Used within WHERE statement to consider more than one logical clause at a time
    - Each time you use statement with AND, you will need to specify the column you are interested in looking at
    - LIKE, IN, NOT can be linked together using AND operator
  + BETWEEN operator can make a cleaner statement in code
    - BETWEEN is inclusive of endpoints!!
    - Instead of writing :

WHERE column >= 6 AND column <= 10

* + - we can instead write, equivalently:

WHERE column BETWEEN 6 AND 10

* **OR** This allows you to combine operations where at least one of the combined conditions must be true.
  + Logical operator that allows you to select rows that satisfies either of two conditions
  + Will need to specify the column you are interested in looking at
  + Can be combined with other operators by using ()

**RECAP:**

| **Statement** | **How to Use It** | **Other Details** |
| --- | --- | --- |
| SELECT | SELECT **Col1**, **Col2**, ... | Provide the columns you want |
| FROM | FROM **Table** | Provide the table where the columns exist |
| LIMIT | LIMIT **10** | Limits based number of rows returned |
| ORDER BY | ORDER BY **Col** | Orders table based on the column. Used with **DESC**. |
| WHERE | WHERE **Col > 5** | A conditional statement to filter your results |
| LIKE | WHERE **Col LIKE '%me%'** | Only pulls rows where column has 'me' within the text |
| IN | WHERE **Col IN ('Y', 'N')** | A filter for only rows with column of 'Y' or 'N' |
| NOT | WHERE **Col NOT IN ('Y', 'N')** | **NOT** is frequently used with **LIKE** and **IN** |
| AND | WHERE **Col1 > 5 AND Col2 < 3** | Filter rows where two or more conditions must be true |
| OR | WHERE **Col1 > 5 OR Col2 < 3** | Filter rows where at least one condition must be true |
| BETWEEN | WHERE **Col BETWEEN 3 AND 5** | Often easier syntax than using an **AND** |

**Other Tips**

Though SQL is **not case sensitive** (it doesn't care if you write your statements as all uppercase or lowercase), we discussed some best practices. **The order of the key words does matter!** Using what you know so far, you will want to write your statements as:

**SELECT** col1, col2

**FROM** table1

**WHERE** col3 > 5 **AND** col4 **LIKE** '%os%'

**ORDER** **BY** col5

**LIMIT** 10;

Notice, you can retrieve different columns than those being used in the **ORDER BY** and **WHERE** statements. Assuming all of these column names existed in this way (col1, col2, col3, col4, col5) within a table called table1, this query would run just fine.