### **The Last SQL Guide for Data Analysis**

By 2020, it’s estimated that 1.7MB of data will be created every second for every person on earth. Absurd! That’s a lot of data to be managed and stored. Luckily, we have ways to handle this growing generated data.A common way to manage this data is through a relational database which stores data in a tabular form consisting of rows and columns. A database consists of both data and metadata. Metadata is the data that describes the data’s structure within the database.

Photo by [Ruchindra Gunasekara](https://unsplash.com/@ruchindra?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com?utm_source=medium&utm_medium=referral)

### **Table of Contents**

* Introduction
* Database Setup
* cheat Sheet
* [B](https://medium.com/p/1c96572a1401#0374)asic retrieval
* Filters
* [J](https://medium.com/p/1c96572a1401#0cd4)oins
* SQL aggregate functions
* [D](https://medium.com/p/1c96572a1401#cdde)atabase transactions
* [D](https://medium.com/p/1c96572a1401#45f6)atabase triggers
* Index
* Tips
* Conclusion

### **Introduction**

My objective in this article is to get you comfortable with SQL and cover most of the concepts individually when coming up with queries in your day to day work life. To be able to communicate with the database directly we use SQL which is an abbreviation for Structured Query Language. SQL is used to perform tasks such as creating,reading,updating and deleting tables in a database.

SQL is a **declarative** and domain specific mostly used by Business Analysts, Software Engineers, Data Analysts and many other professions that make use of gathered information in form of data. You don’t need to be a programmer or know programming languages such as Python to master SQL. SQL’s syntax is similar to English . With a little bit of memorization of the simple syntax you are ready to work comfortably with database systems.

“Declarative programming is where you say what you want without having to say how to do it. With procedural programming, you have to specify exact steps to get the result.” — Stack Overflow

### **Database Setup**

We will be using a test database to write our own personal queries and use W3school’s SQL editor found [here](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_all). It requires no installation so we can just focus on writing queries.

### **Cheat Sheet**

Since SQL is declarative It will help us tremendously if we just memorize the SQL statements. Having a reference guide near you at all times will help you quickly grasp the keywords used in querying the tables. [Here](https://docs.google.com/spreadsheets/d/1_HkCVTOpkaDPzg4dmFLJRYQhODKiVBRdX7F3GUI6O3s/edit?usp=sharing) is a spreadsheet of keywords that I made for your references.

### **Basic Retrieval**

Most of our queries will involve some form of action to be performed on a table which falls under four tasks creating,inserting,updating and deleting tables.

#### **Create a table**

When we want to create tables in a database we use the **create table** statement. Type in the following code in the editor.

CREATE TABLE Countries(

Country\_id int,

Country\_name varchar(255),

Continent varchar(255),

Population int

);

What this does is create a table named countries which has four columns with its respective data types. The minimum that is required to make a table in SQL is the name of the column,the data types and length, you can have more characteristics such Not Null meaning the row wont be entered in the database if it contains an empty data.

#### **Insert a table**

After creating a table we can insert rows into the tables by using the **insert into** method statement. Type in the following code

INSERT INTO countries(Country\_id,Country\_name, Continent,Population)

VALUES (1,'Somalia','Africa',14000000);

This statement adds Somali as a new country into the the countries table. It is a good measure to specify the column names and values when inserting rows into the table.

#### **Read a table**

In case we want to look up the data that we stored in the database, we use the **Select** statement.

Select \* from Countries;

This statement returns a table that displays the row we just inserted with out insert statement. The \* wildcard means show me **“all”** the column and its values in the table. If you want your table to only display “Population” Column we remove the asterisk and replace it with the column name.

Select Population from Countries;

#### **Update a table**

If we want to modify existing records in a table we use the **update** statement to do so.

UPDATE Countries

SET Country\_name ='Kenya'

WHERE Country\_id=1;

This statement updates the country name of the country id 1 to “Kenya”. We have to specify which country id because we only want to change that row. If we remove the where statement, SQL will assume that you want to update all the rows in the table.

#### **Delete records in a table**

If we want to delete all rows in a table, we would then use **delete from** statement.

DELETE FROM Countries;

If you want to delete the table instead of all records, we use the **drop table** statement.

DROP TABLE Countries;

**Note:** This removes the whole table from the database and can result in loss of data!

### **Filters**

If we are interested in only part of the data in the table we can filter the table. We have multiple statements that filters our tables by selecting certain rows of the table that match a certain criteria and returning the results back as a data set.

#### **WHERE**

The where clause is used to filter records. In our editor we have a table called customers. If we want to filter customers that are from “USA” we use the where statement.

SELECT \* from Customers WHERE country = "USA";

#### **AND/OR/Not**

In our previous, example we had only one condition which was “where country is USA” we could combine multiple conditions using and/or/not. For example if you want customers from USA or Brazil. you use the **or** statement.

SELECT \* from Customers WHERE country = "USA" OR country = "Brazil";

#### **ORDER BY**

Most of the times when we filter the table, the data set we get back is unsorted . We can sort this filtered unsorted data set using an **order by** statement.

SELECT \* from Customers WHERE country = "USA" OR country = "Brazil"

ORDER BY CustomerName ASC;

This will order the filtered results alphabetically . If we want to sort it in descending manner we replace ASC with DESC.

#### **BETWEEN**

sometimes we would like filter rows of a specific range. We use the **between** statement to select range.

SELECT \* from Products

WHERE Price BETWEEN 10 AND 20;

The statement above filters products whose price falls between 10 and 20.

**Note:** In a Between operation, the lower bound and upper bound are both inclusive in nature.

#### **LIKE**

Sometimes we want to filter the table with a specific pattern in mind, we use the **like** statement.

SELECT \* from Customers

WHERE CustomerName LIKE 'A%';

The SQL statement above filters the table to show only customers whose name begin with the letter A. If you bring the percentage sign forward it would mean customers whose name ends with the letter A.

#### **GROUP BY**

**Group by** mostly just groups the filtered result set into groups. Think of it like a summary for each column data set.

SELECT COUNT(CustomerID), Country

FROM Customers

GROUP BY Country;

This statement counts the number of customers from each country then groups it into countries. Group by method is mostly used with aggregate functions which we will talk about in details further down the article.

#### **HAVING**

Having was introduced because the **where** statement does not work with aggregate functions because it only deals with direct values in the database.

SELECT COUNT(CustomerID), Country

FROM Customers

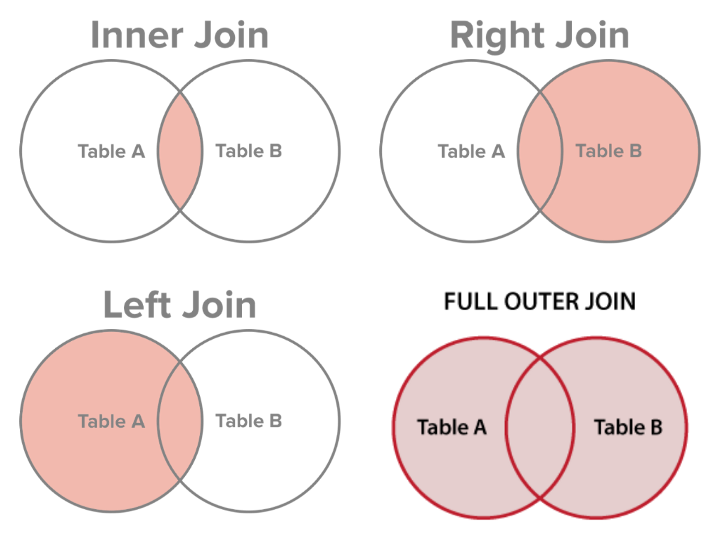
GROUP BY Country

HAVING COUNT(CustomerID) > 3;

This statement does the same thing as last example only difference is we only include countries that have more than three customers

### **Joins**

Imagine you’d like to know which customer ordered a what products. If a database follows a proper database normalization techniques then products,customers and orders would be in separate tables . If we want to see which customer ordered what products, we would then have to look at the customer id inside the order table then go to the customer table and see the products he bought using the product id in the product table. As you can see this is a huge headache if were to repeat it multiple times, in order to do this easier SQL has come up with a statement called **joins.**This clause is used to combine two or more rows of tables based on a shared related column.



#### **Inner Join**

Inner join commonly known as only **join** is used to merge related tables at a shared column into a single table.

SELECT Orders.OrderID, Customers.CustomerName

FROM Orders

INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;

The statement above returns the columns order id and customer names. We join these columns from orders and customers only those that are found in both customer and order tables.

#### **Left Join**

SELECT Customers.CustomerName, Orders.OrderID

FROM Customers

LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID

ORDER BY Customers.CustomerName;

In the left join statement we join the left table(customer) and right table(order) this returns all rows from the left table and matched records from the right table.

#### **Right Join**

SELECT Orders.OrderID, Employees.LastName, Employees.FirstName

FROM Orders

RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID

ORDER BY Orders.OrderID;

The right join returns all rows from the right table and matched records from the left table. This returns all employees and any order the might have placed.

#### **Outer Join**

SELECT Customers.CustomerName, Orders.OrderID

FROM Customers

FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID

ORDER BY Customers.CustomerName;

Also known as **full outer join** and is used to combine all rows from one or more tables. No rows will be left out, all will be included in the joined tables.

### **SQL aggregate functions**

A function is a set of procedures that take an input and spits out a result. It is basically a set of SQL statements.There are two types of functions in SQL **set functions** and **value functions**.Any function that manipulates rows of data in a table and returns a single value is called an set function and programmers call it aggregate functions because it takes rows in the table and returns a singular aggregate information.

#### **MIN**

SELECT MIN(Price) AS LeastPricy

FROM Products;

This returns the cheapest price of all the products available in the products table.

#### **MAX**

SELECT MAX(Price) AS MostExpensive

FROM Products;

This returns the most expensive price of all the products available in the products table.

#### **AVG**

SELECT AVG(Price) AS AveragePrice

FROM Products;

This returns the average price of all the products available in the products table.

#### **COUNT**

SELECT COUNT(ProductID)

FROM Products;

This returns the number of products available in the products table.

#### **SUM**

SELECT SUM(Quantity)

FROM OrderDetails;

This returns the sum of all the orders in the orderDetails table.

### **Indexes**

All the queries we had were basic queries but most of the times we usually execute a combination of multiple SQL statements which if it gets complex enough will lower the execution time. Luckily, SQL has something called **indexing** which allows for faster Look up time for scanning data. An index is basically a data structure that has a pointer to the data in a table that is sorted. Without an index, the searching data in a table would be **linear** meaning it would be like going through one row after the other.Index is well suited for tabular data.

#### **Creating an Index**

CREATE INDEX idx\_lastname

ON Persons (LastName);

This will create an index to look up data from the column quickly. It is to be noted that indexes are **not** stored in the table and are invisible. We most often use indexes when we have tables that users search through a lot.

### **Database transactions**

Transactions is a collection of a couple of SQL statements that must be executed for it to be a successful operation. Transactions are all or nothing kind of operations. If all but one operations fail we consider that transaction to fail.

A common example when we use transactions is when we transfer money from one account to another in a bank. In order for a transfer to be successful money must be removed from account A and added to account B for it to be a successful transaction otherwise we would roll back the transaction to start a fresh. When transaction complete we say that the transaction is committed This ensures the database to maintain data integrity and data consistency.

If you want to learn in depth into what database transaction are then I highly suggest you take a look at [This](https://www.youtube.com/watch?v=is03uRYFgqc) excellent YouTube video explanation of database transactions.

### **Database triggers**

Sometimes not all SQL queries are individual and isolated. We would like to do something to another table A when a separate event happens in another table B. This is where we get to use a **database trigger.**

A **database trigger** is a bunch of SQL statements that are run when a specific actions occur within a database. Most triggers are defined to run when changes are made to a table’s data. Triggers are mostly run before or after actions such as delete, update and create. Most common use cases of database triggers is validating input data

### **Tips**

1. all SQL reserved words are uppercase, all identifier such as tables and columns are lowercase.
2. Divide your queries into multiple lines instead of one long statement in one line.
3. You cannot add a column at a specific position in a table so be cautious when designing the tables.
4. Be aware when using AS alias, the columns are not being renamed in the table. The aliases only appear in the results.
5. SQL evaluates these clauses in the order from, where, group by, having, and finally select. Therefore each clause receives the filtered results of the previous filter. It would look like this.

SELECT(HAVING(GROUP BY(WHERE(FROM...))))

#### **Conclusion**

You can generate powerful queries from a permutation of the the SQL statements we saw above. The best way to cement the concepts and get better at SQL is practicing and solving problems. You can find interactive exercises in websites like [hackerrank](http://hackerrank.com) and [LeetCode](http://leetcode.com).

**The more you practice the better you’ll be, the harder you train the great in you they’ll see.***— Alcurtis Turner*

Wishing you peace and prosperity!

### 