Chapter 1 : Basic SQL

Aviral Janveja

What is SQL?

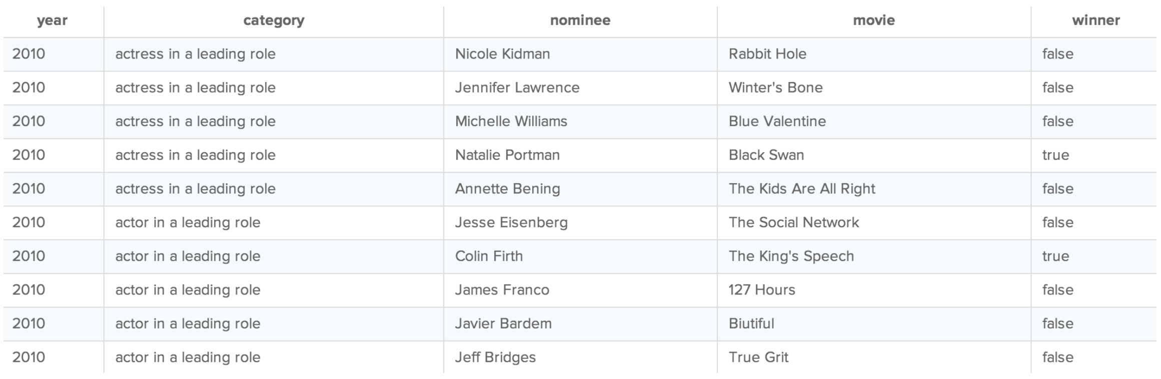
Structured Query Language is the standard language for querying, storing, manipulating and retrieving data from relational databases. A query is an inquiry or request for data from a database.

What is a Database?

A database is an organized collection of data that is stored digitally, such that information can be maintained, accessed and analyzed efficiently. In order to manage our databases, we use a software called **DBMS**, short for Database Management System.

What is a Relational Database ?

A relational database is a type of database that organizes data into rows and columns, which collectively form a table where the data points are related to each other. If you've used Excel, you should already be familiar with tables. Relational Database tables have rows and columns just like Excel, but are a little more rigid. For instance, they are always organized by columns and each column must have a unique name. To get a sense of this organization, the image below shows a sample table.



year	category	nominee	movie	winner
2010	actress in a leading role	Nicole Kidman	Rabbit Hole	false
2010	actress in a leading role	Jennifer Lawrence	Winter's Bone	false
2010	actress in a leading role	Michelle Williams	Blue Valentine	false
2010	actress in a leading role	Natalie Portman	Black Swan	true
2010	actress in a leading role	Annette Bening	The Kids Are All Right	false
2010	actor in a leading role	Jesse Eisenberg	The Social Network	false
2010	actor in a leading role	Colin Firth	The King's Speech	true
2010	actor in a leading role	James Franco	127 Hours	false
2010	actor in a leading role	Javier Bardem	Biutiful	false
2010	actor in a leading role	Jeff Bridges	True Grit	false

Figure 1: Example Table

The SQL queries learned here can be practiced on real world data at **mode.com**.

1 SELECT and FROM

Let us start by looking at some basic SQL syntax and queries. There are two required ingredients in any SQL query : **SELECT** and **FROM** and they have to be in that order. SELECT indicates the columns that you would like to view and FROM identifies the table that they are in.

```
SELECT year,
       month,
       west
    FROM tutorial.us_housing_units
```

The query above is telling the database to return the year, month and west columns from the "tutorial.us_housing_units" table.

1.1 Formatting Conventions

You might have noticed that the SELECT and FROM commands are capitalized. This isn't actually necessary. SQL will understand these commands if you type them in lowercase. Capitalizing commands is simply a convention that makes queries easier to read. Similarly, SQL treats one space, multiple spaces or a line break as being the same thing.

Note that the three column names were separated by a comma in the query. Whenever you select multiple columns, they must be separated by commas, but you should not include a comma after the last column name.

1.2 Column Names

While we are on the topic of formatting, it is worth noting the format of column names. All of the columns in the tutorial.us_housing_units table are named in lower case, and use underscores instead of spaces. The table name itself also uses underscores instead of spaces.

If you want to have spaces in column names, you need to always refer to those columns in double quotes. For example, if you want the "west" column to appear as "West Region" in the results, you would have to type:

```
SELECT west AS "West Region"
    FROM tutorial.us_housing_units
```

2 WHERE

Once you know how to view some data using SELECT and FROM, the next step is filtering the data using the WHERE clause. Here's what it looks like :

```
SELECT *
    FROM tutorial.us_housing_units
    WHERE month = 1
```

The WHERE clause works in a plain english way, which means that the results will only include rows where the "month" column contains the value 1. Also **note** that the clauses always need to be in this order : first SELECT, then FROM and then WHERE.

3 LIMIT

Limit restricts how many rows the SQL query returns. Many analysts use limit as a simple way to keep their queries from taking too long to return, especially when they only need the first few rows to see what a particular table looks like. This is how you can add limit to a SQL command :

```
SELECT *
    FROM tutorial.us_housing_units
    LIMIT 15
```

4 Comparison Operators

Comparison operators in SQL include Equal to (=), Not equal to (! =), Greater than (>), Less than (<), Greater than or equal to (>=), Less than or equal to (<=). The most basic way to filter data is using comparison operators. For example :

```
SELECT *
    FROM tutorial.us_housing_units
    WHERE west > 30
```

These comparison operators make the most sense when applied to numerical columns. However, all of the above operators work on non-numerical data as well. There are some important rules though. If you are using an operator with values that are non-numeric, you need to put the value in single quotes, for example :

```
SELECT *
    FROM tutorial.us_housing_units
    WHERE month_name = 'January'
```

Comparison operators on non-numeric columns filter based on alphabetical order.

5 Arithmetic Operators

You can perform arithmetic in SQL using the same common operators $+$, $-$, $*$, $/$. You can use parentheses to manage the order of operations. It makes sense to use parentheses even when it is not absolutely necessary, just to make your query easier to read.

```
SELECT year,
   month,
   west,
   south,
   west + south AS south_plus_west
    FROM tutorial.us_housing_units
```

The columns that contain the arithmetic functions are called "derived columns" because they are generated by modifying the information that exists in the underlying table.

6 Logical Operators

Logical operators allow you to use multiple comparison operators in one query. We will go through each one of them individually in the following sections.

6.1 LIKE

LIKE is a logical operator in SQL that allows you to match on similar values rather than exact ones. For example :

```
SELECT *
    FROM tutorial.billboard_top_100_year_end
    WHERE "group_name" LIKE 'Snoop%'
```

In this example, the results will include rows for which "group_name" starts with "Snoop" and is followed by any number and selection of characters. The "%" used above represents any character or set of characters. LIKE is case-sensitive, meaning that the above query will only capture matches that start with a capital "S" and lower-case "noop". To ignore case when you are matching values, you can use the **ILIKE** command :

```
SELECT *
    FROM tutorial.billboard_top_100_year_end
    WHERE "group_name" ILIKE 'snoop%'
```

You can also use "_" (a single underscore) to substitute for an individual character :

```
SELECT *
    FROM tutorial.billboard_top_100_year_end
    WHERE artist ILIKE 'dr_ke'
```

6.2 IN

IN is a logical operator in SQL that allows you to specify a list of values that you would like to include in the results. For example, the following query of data will return results for which the year_rank column is equal to one of the values in the list :

```
SELECT *
    FROM tutorial.billboard_top_100_year_end
    WHERE year_rank IN (1, 2, 3)
```

As with comparison operators, you can use non-numerical values, but they need to go inside single quotes. Regardless of the data type, the values in the list must be separated by commas.

6.3 IS NULL

Some tables contain null values : cells with no data in them at all. You can select rows that contain no data in a given column by using IS NULL :

```
SELECT *
    FROM tutorial.billboard_top_100_year_end
    WHERE artist IS NULL
```

"WHERE artist = NULL" will not work here as you can't perform arithmetic on null values.

6.4 AND

AND is a logical operator in SQL that allows you to select those rows that satisfy both of the two specified conditions. For example :

```
SELECT *
    FROM tutorial.billboard_top_100_year_end
    WHERE year_rank >= 5 AND year_rank <= 10
```

You can use SQL's AND operator with additional AND statements or any other comparison operator, as many times as you want. If you run the query below, you will notice that all of the requirements are satisfied.

```
SELECT *
    FROM tutorial.billboard_top_100_year_end
    WHERE year = 2012
    AND year_rank <= 10
    AND "group_name" ILIKE '%feat%'
```

You can see that the above query is spaced out onto multiple lines, a good way to make long WHERE clauses more readable.

6.5 OR

OR is a logical operator in SQL that allows you to select rows that satisfy either of the two specified conditions. For example :

```
SELECT *
    FROM tutorial.billboard_top_100_year_end
    WHERE year_rank = 5 OR artist = 'Gotye'
```

You can combine AND with OR using parenthesis. The following query will return rows that satisfy both of the following conditions :

```
SELECT *
    FROM tutorial.billboard_top_100_year_end
     WHERE year = 2013
     AND ("group_name" ILIKE '%macklemore%' OR
    "group_name" ILIKE '%timberlake%')
```

6.6 NOT

NOT is a logical operator in SQL that you can put before any conditional statement to select rows for which that statement is false.

```
SELECT *
    FROM tutorial.billboard_top_100_year_end
    WHERE NOT year_rank = 1
```

In the above case, you can see that results for which year_rank equal to 1 are not included. NOT is commonly used with LIKE, for example :

```
SELECT *
    FROM tutorial.billboard_top_100_year_end
    WHERE year = 2013
    AND "group_name" NOT ILIKE '%macklemore%'
```

NOT is also frequently used to identify non-null rows, but you need to include IS beforehand. Here's how that looks :

```
SELECT *
    FROM tutorial.billboard_top_100_year_end
    WHERE year = 2013
    AND artist IS NOT NULL
```

6.7 BETWEEN

BETWEEN is a logical operator in SQL that allows you to select only rows that are within a specific range, **including** the range bounds that you specify in the query. It has to be paired with the AND operator :

```
SELECT *
    FROM tutorial.billboard_top_100_year_end
    WHERE year = 2013
    AND year_rank BETWEEN 2 AND 4
```

In the above case, you can see the results for the year 2013, where year_rank is equal to 2, 3 or 4.

7 ORDER BY

Once you have learned how to filter data, it is time to learn how to sort data. The ORDER BY clause allows you to reorder your results based on the data in one or more columns. By default, SQL sorts in ascending order.

```
SELECT *
    FROM tutorial.billboard_top_100_year_end
    WHERE year = 2013
    ORDER BY year_rank
```

If you would like your results in the descending order, you need to add the **DESC** operator :

```
SELECT *
    FROM tutorial.billboard_top_100_year_end
    WHERE year = 2013
    ORDER BY year_rank DESC
```

You can also order by multiple columns :

```
SELECT *
    FROM tutorial.billboard_top_100_year_end
    WHERE year_rank <= 3
    ORDER BY year DESC, year_rank
```

You can **note** a couple things from the above query. First, columns in the ORDER BY clause must be separated by commas. Second, the DESC operator is only applied to the column that precedes it. Finally, the results are sorted by the first column mentioned, then the second one and so on.

8 Comments

You can comment out pieces of code. In other words, you can specify parts of your query that will not actually be treated like SQL code. It can be helpful to include comments that explain your thinking so that you can easily remember what you intended to do if you ever wish to revisit your work.

Commenting can also be useful if you want to test variations on your query while keeping all of your code intact. You can use −− (two dashes) to comment out everything to the right on a given line :

```
SELECT *
-- This comment won't affect the way the code runs
    FROM tutorial.billboard_top_100_year_end
    WHERE year = 2013
```

You can also leave comments across multiple lines using /* to begin the comment and */ to close it :

```
/* Here's a comment so long and descriptive that
it could only fit on multiple lines. Fortunately,
it, too, will not affect how this code runs. */
SELECT *
    FROM tutorial.billboard_top_100_year_end
    WHERE year = 2013
```

Chapter End.