@Auther: Gabir N. Yusuf
@Date: 25 of Aug 2019
@Quizes:https://github.com/gabir-yusuf/100DaysOfDataScience/tree/master/1_Basic_SQL

Entity Relationship Diagrams

An **entity-relation shipiagram** (ERD) is a common way to view data in a database.

SQL vs. NoSQL

NoSQL stands for not only SQL. Databases using NoSQL allow for you to write code that interacts with the data a bit differently than what we will do in this course. These NoSQL environments tend to be particularly popular for web based data, but less popular for data that lives in spreadsheets the way we have been analyzing data up to this point. One of the most popular NoSQL languages is called MongoDB.

A few key points about data stored in SQL databases:

1. **Data in databases is stored in tables that can be thought of just like Excel spreadsheets.**
2. **All the data in the same column must match in terms of data type.**
3. **Consistent column types are one of the main reasons for working with databases is fast.**

Types of Databases

SQL Databases

There are many different types of SQL databases designed for different purposes. In this course, we will use **Postgres** within the classroom, which is a popular open-source database with a very complete library of analytical functions. Some of the most popular databases include:

1. MySQL
2. Access
3. Oracle
4. Microsoft SQL Server
5. Postgres

You can also write SQL within other programming frameworks like Python, Scala, and HaDoop.

The key to SQL is understanding **statements**. A few statements include:

1. **CREATE TABLE** is a statement that creates a new table in a database.
2. **DROP TABLE** is a statement that removes a table in a database.
3. **SELECT** allows you to read data and display it. This is called a **query**.

The **SELECT** statement is the common statement used by analysts.

SQL statements are code that can read and manipulate data. Basic syntax reminders: **SQL isn't case sensitive**. Additionally, you can end SQL statements with a semicolon, but some SQL environments don't require a semicolon at the end.

Here you were introduced to the SQL command that will be used in every query you write: SELECT ... FROM ....

1. **SELECT** indicates which column(s) you want to be given the data for.

2. **FROM** specifies from which table(s) you want to select the columns. Notice the columns need to exist in this table.

If you want to be provided with the data from all columns in the table, you use "*", like so:

```
SELECT *
FROM orders
```

Note that using SELECT does not *create* a new table with these columns in the database, it just provides the data to you as the results, or output, of this command.

SELECT and FROM in Every SQL Query

- Every query will have at least a **SELECT** and **FROM** statement. The **SELECT** statement is where you put the **columns** for which you would like to show the data.
- The **FROM** statement is where you put the **tables** from which you would like to pull data.

To show these three columns (id, account_id, occured_at) from order query:

```
SELECT id, account_id, occurred_at

FROM orders;
```

Using Upper and Lower Case in SQL

SQL queries are not case-sensitive. **But,** it is common and best practice to capitalize all SQL commands, like SELECT and FROM, and keep everything else in your query lower case.

One other note: The text data stored in SQL tables can be either upper or lower case, and SQL *is* case-sensitive in regard to this text data.

Use White Space in Queries

SQL queries ignore spaces, so you can add as many spaces and blank lines between code as you want, and the queries are the same. This query

```
SELECT account_id FROM orders
```

is equivalent to this query:

```
SELECT account_id

FROM orders
```

Semicolons

Depending on your SQL environment, your query may need a semicolon at the end to execute. Other environments are more flexible in terms of this being a "requirement." It

is considered a best practice to put a semicolon at the end of each statement, which also allows you to run multiple queries at once if your environment allows this.

LIMIT Statement

- The LIMIT statement is useful when you want to see just the first few rows of a table. This can be much faster for loading than if we load the entire dataset.
- The LIMIT command is always the very last part of a query.

ORDER BY statement

allows us to sort our results using the data in any column. that using ORDER BY in a SQL query only has temporary effects, for the results of that query

The **ORDER BY** statement always comes in a query after the **SELECT** and **FROM** statements, but before the **LIMIT** statement. If you are using the **LIMIT** statement, it will always appear last. As you learn additional commands, the order of these statements will matter more.

Pro Tips

- Remember `DESC` can be added after the column in your ORDER BY statement to sort in descending order, as the default is to sort in ascending order.
- we can **ORDER BY** more than one column at a time. When you provide a list of columns in an **ORDER BY** command, the sorting occurs using the leftmost column in your list first, then the next column from the left, and so on. We still have the ability to flip the way we order using **DESC**.

WHERE statement

- Using the **WHERE** statement, we can display *subsets* of tables based on conditions that must be met.
- The **WHERE** statement can also be used with non-numeric data. We can use the `=` and `!=` operators here.
- Commonly when we are using **WHERE** with non-numeric data fields, we use the **LIKE**, **NOT**, or **IN** operators.

Derived Columns

Creating a new column that is a combination of existing columns is known as a **derived column**. Usually you want to give a name, or "alias," to your new column using the **AS** keyword.

This derived column, and its alias, are generally only temporary, existing just for the duration of your query.

You can use mathematical expressions like addition, subtraction, multiplication, and division.

Consider this example:

```
SELECT id, (standard_amt_usd/total_amt_usd)*100 AS std_percent, total_amt_usd

FROM orders

LIMIT 10;
```

The **AS** keyword to name this new column "std_percent."

Introduction to Logical Operators

1. **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.

2. **IN**

   This allows you to perform operations similar to using **WHERE** and `=`, but for more than one condition.

3. **NOT**

   This is used with **IN** and **LIKE** to select all of the rows **NOT LIKE** or **NOT IN** a certain condition.

4. **AND & BETWEEN**

   These allow you to combine operations where all combined conditions must be true.

5. **OR**

   This allows you to combine operations where at least one of the combined conditions must be true.

LIKE Operator

extremely useful for working with text and used within a **WHERE** clause.

The **LIKE** operator is frequently used with `%`. The `%` 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.

Remember you will need to use single quotes for the text you pass to the **LIKE** operator, because of this lower and uppercase letters are not the same within the string. Searching for **'T'** is not the same as searching for **'t'**.

IN operator

Useful for working with both numeric and text columns.

This operator allows you to use an `=`, but for more than one item of that particular column. We can check one, two or many column values for which we want to pull data, but all within the same query. In the upcoming concepts, you will see the **OR** operator that would also allow us to perform these tasks, but the **IN** operator is a cleaner way to write these queries.

NOT Operator

is an extremely useful operator for working with the previous two operators we introduced: **IN** and **LIKE**. By specifying **NOT LIKE** or **NOT IN**, we can grab all of the rows that do not meet particular criteria.

AND Operator

- Used within a **WHERE** statement to consider more than one logical clause at a time.
- Each time you link a new statement with an **AND**, you will need to specify the column you are interested in looking at.
- You may link as many statements as you would like to consider at the same time.
- This operator works with all of the operations we have seen so far including arithmetic operators (`+`, `*`, `-`, `/`). **LIKE**, **IN**, and **NOT** logic can also be linked together using the **AND** operator.

BETWEEN Operator

Sometimes we can make a cleaner statement using **BETWEEN** than we can using **AND**. Particularly this is true when we are using the same column for different parts of our **AND** statement.

OR Operator

- It can combine multiple statements.
- Each time you link a new statement with an **OR**, you will need to specify the column you are interested in looking at.
- You may link as many statements as you would like to consider at the same time. This operator works with all of the operations we have seen so far including

arithmetic operators (`+`, `*`, `-`, `/`), **LIKE**, **IN**, **NOT**, **AND**, and **BETWEEN** logic can all be linked together using the **OR** operator.

- When combining multiple of these operations, we frequently might need to use parentheses to assure that logic we want to perform is being executed correctly. The video below shows an example of one of these situations.

Recap

Commands

You have already learned a lot about writing code in SQL! Let's take a moment to recap all that we have covered before moving on:

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 the 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.

Go to GitHub to see the solution of quizzes on this lesson:

https://github.com/gabir-yusuf/100DaysOfDataScience/tree/master/1_Basic_SQL