More about SQL in action

Just as humans use different languages to communicate with others, so do computers. Structured Query Language (or SQL, sometimes pronounced as “sequel”) lets data analysts talk to their databases. SQL is one of the most useful data analyst tools, especially when you are working with large datasets in tables. It can help you investigate huge databases, track down text (referred to as strings) and numbers, and filter for the exact kind of data you need—much faster than a spreadsheet can. In this reading, we will go over the basics of using SQL and explore an example query to demonstrate how SQL works in action.

Basic syntax for SQL

Every programming language, including SQL, has to follow a unique set of guidelines known as **syntax**. As soon as you enter your search criteria using the correct syntax, the query should start working to pull the information you have requested from the target database.

What is a query?

A **query** is basically a request for data or information from a database. For example, "Tell me how many comedy movies were made in 1985” or "How many people live in Puerto Rico?" When we query databases, we use SQL to communicate our question or request. Both the user and the database can always exchange information as long as you “speak” the same language.

The foundation of every SQL query is the same:

- Use SELECT to choose the columns you want to return.
- Use FROM to choose the tables where the columns you want are located.
- Use WHERE to filter for certain information.

Basic components of a query (and a few useful tips)

```
SELECT
      field1
FROM
      table
WHERE
      field1 = condition;
```

Tip 1: You can write your SQL queries all in lowercase and don’t have to worry about proper spacing between words. With that said, however, using capitalization and spacing can help you read the information more easily. Keep your queries neat, and they will be easier to review or troubleshoot if you need to check them later on.

In the SQL syntax shown above, the SELECT statement identifies the column you want to pull data from by name, field1, and the FROM statement identifies the table in which the column is located by name, table. Finally, the WHERE statement narrows your query so that the database returns only the data with an exact value match or the data that matches a certain condition that you want.

For example, if you are looking for a specific customer with the last name Chavez, the **WHERE** statement would be: **WHERE field1 = 'Chavez'**;

However, if you are looking for all customers with a last name that begins with the letters "Ch", the **WHERE** statement would be: **WHERE field1 LIKE 'Ch%'**;

You can see that the LIKE statement is very powerful because it allows you to tell the database to look for a certain pattern! The percent sign (%) is used as a wildcard to match one or more characters. In our example, both Chavez and Chen would be returned. Note that in some databases the asterisk (*) is used as the wildcard instead of the percent sign (%).

Can you use SELECT * ?

In our example, if you replace **SELECT field1** with **SELECT *** you would be selecting all the columns in the table. From a syntax point of view, it is a correct SQL statement, but you should use it sparingly and with caution because depending on how many columns a table has, you could be selecting a tremendous amount of data.

Finally, you will notice that we have shown the SQL statement with a semicolon at the end. The semicolon is a statement terminator and is part of the American National Standards Institute (ANSI) SQL-92 standard which is a recommended common syntax for adoption by all SQL databases. However, not all SQL databases have adopted or enforce the semicolon, so it's possible you may come across some SQL statements that aren't terminated with a semicolon. If a statement works without a semicolon, it's fine.

Tip 2: Some tables aren't designed with descriptive enough naming conventions. In our previous example, field1 was the column for a customer's last name, but you wouldn't know it by the name. A better name would have been something like last_name. In these cases, you can place comments alongside your SQL statements to help you remember what the name represents. Comments are text placed between certain characters, **/*** and ***/**, or after two dashes (**--**) as shown below.

```
SELECT
      field1 /* this is the last name column */
FROM
      table -- this is the customer data table
WHERE
      field1 LIKE 'Ch%';
```

Tip 3: You can also make it easier on yourself by assigning a new name or **alias** to the column or table names to make them easier to work with (and avoid the need for comments). This is done with a SQL **AS** statement. In the example below, you are changing field1 to last_name and table to customers for the duration of the query only. It doesn't change the names in the actual table used in the database.

```
field1 AS last_name -- Alias to make my work easier
table AS customers -- Alias to make my work easier

SELECT
      last_name
FROM
      customers
WHERE
      last_name LIKE 'Ch%';
```

Putting SQL to work (what you might do as a data analyst)

Imagine you are a data analyst for a small business and your manager asks you for some employee data. You decide to write a query with SQL to get what you need from the database.

Let's say you want to pull all the columns: empID, firstName, lastName, jobCode, and salary. Because you know the database isn't that big, instead of creating a SELECT statement for each column, you use SELECT *. This will select all the columns from the Employee table in the FROM statement.

```
SELECT
      *
FROM
      Employee
```

Now, let's get more specific about the data we want from the Employee table. If you want all the data about employees working in the SFI job code, you can use a WHERE statement to filter out the data based on this additional requirement.

Here, you use:

```
SELECT
      *
FROM
      Employee
WHERE
      jobCode = 'SFI'
```

The portion of the resulting data returned from the SQL query might look like this:

empID	firstName	lastName	jobCode	salary
0002	Homer	Simpson	SFI	15000
0003	Marge	Simpson	SFI	30000
0034	Bart	Simpson	SFI	25000
0067	Lisa	Simpson	SFI	38000
0088	Ned	Flanders	SFI	42000
0076	Barney	Gumble	SFI	32000

Suppose you notice a large salary range for the SFI job code, so you would like to flag all employees in all departments with lower salaries for your manager. Because interns are also included in the table and they have salaries less than $30,000, you want to make sure your results give you only the full time employees with salaries that are $30,000 or less. In other words, you want to exclude interns with the INT job code who earn less than $30,000. A SQL AND statement will enable you to find this information.

You create a SQL query similar to below, where <> means "does not equal":

```
SELECT
      *
FROM
      Employee
WHERE
      jobCode <> 'INT'
      AND salary <= 30000;
```

The resulting data from the SQL query might look like the following (interns with the job code INT aren't returned):

empID	**firstName**	**lastName**	**jobCode**	**salary**
0002	Homer	Simpson	SFI	15000
0003	Marge	Simpson	SFI	30000
0034	Bart	Simpson	SFI	25000
0108	Edna	Krabappel	TUL	18000
0099	Moe	Szyslak	ANA	28000

With quick access to this kind of data in SQL, you can provide your manager with tons of different insights about employee data, including whether employee salaries across the business are equitable. Fortunately, the query shows only an additional two employees might need a salary adjustment and you share the results with your manager.

Pulling the data, analyzing it, and implementing a solution might ultimately help improve employee satisfaction and loyalty–making SQL a pretty powerful tool.

Resources to learn more

- SQL Cheat Sheet: This starter guide for standard SQL syntax used in PostgreSQL offers videos, activities, and readings on SQL. By the time you are finished, you will get to know a lot more about SQL and be prepared to use it for business analysis and other tasks.
- W3Schools SQL Tutorial: If you would like to explore a detailed tutorial of SQL, this is the perfect place to start. This tutorial includes interactive examples you can edit, test, and recreate. Use it as a reference or complete

the whole tutorial to practice using SQL. Click the green **Start learning SQL now** button or the **Next** button to begin the tutorial.