SELECT to Display Data

by Sophia

WHAT'S COVERED

Asking for more data than you need when building a query can slow down its execution. In this lesson, you will learn how to use SELECT statement modifiers to limit query results to only the columns and rows you need. Specifically, this lesson will cover:

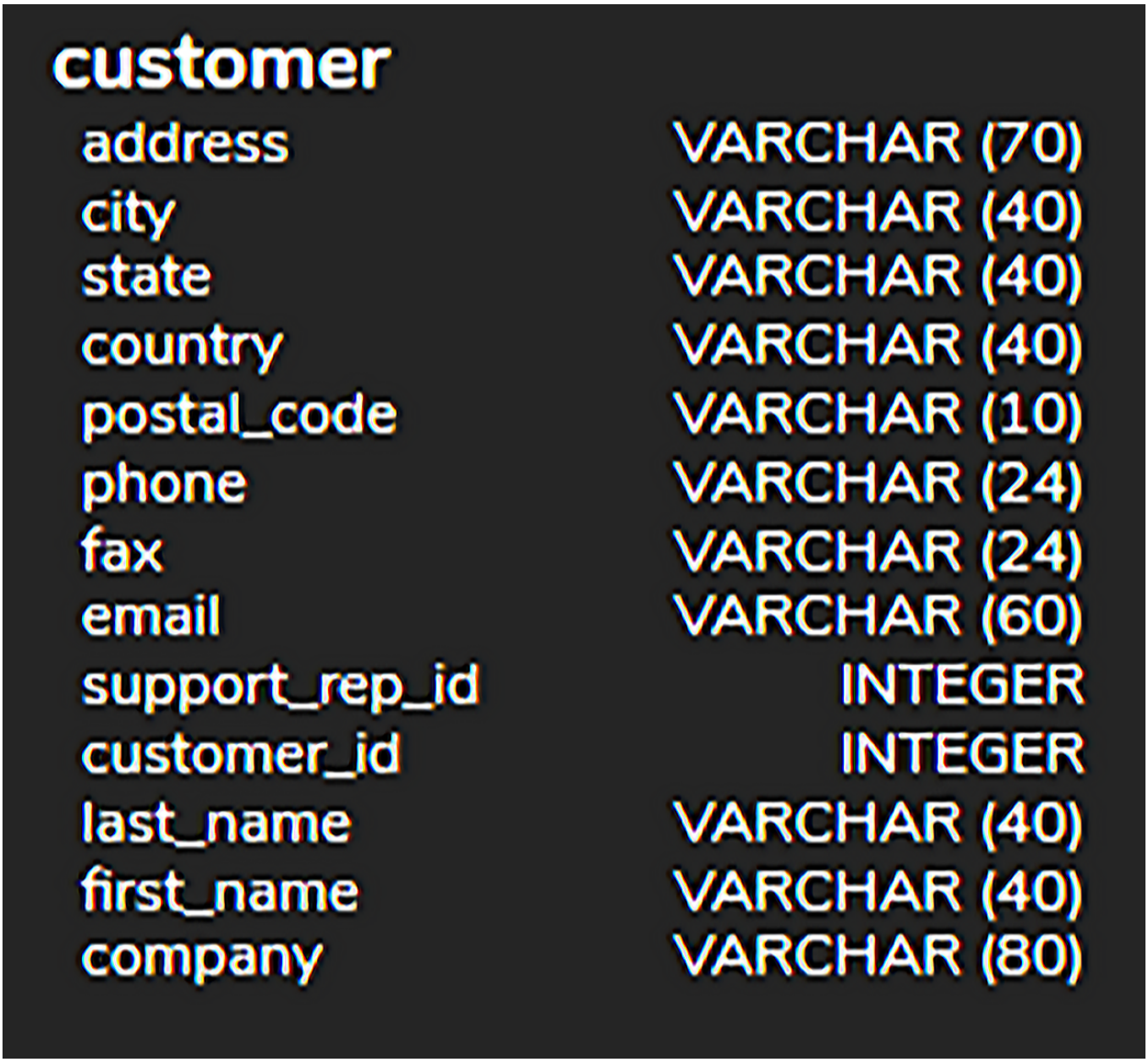
1. Using the SELECT Statement to Display Data

In a previous lesson, you learned how to use SELECT to display data from all columns in a table using the wild card *.

There are instances where you may not need or want to see all the data in a table at once. Retrieving unnecessary data can be expensive in terms of computer power, and for super huge tables or very large databases, this can actually slow down the database, making it run out of processing capacity at the hardware level. Some super large databases will not let you use the * at all because of that. Still, we need a way to find the information we need, and the schema browser provides this capability.

The **schema browser** is a part of the user interface (in this case, PostgreSQL) that shows you all the tables in the database and a list of their attributes (columns). It does not show the data.

The schema browser image shown below contains a list of the attributes in the customer table. Next to each column is its data type. For example, the address attribute's data type is VARCHAR(70), which means it holds variable-length data of up to 70 characters. Notice that the customer_id attribute's data type is INTEGER.



EXAMPLE Sometimes you may need to shape data to make it fit in the database, given a certain attribute's data type. In the customer table, the city attribute has a data type of VARCHAR(40). But what do we do with a city name longer than 40 characters?

Including spaces, Grosse Pointe Shores' official name is “Village of Grosse Pointe Shores, A Michigan City”! The USPS and the city itself seem to use the shorter version of this super long city name: “Grosse Pointe Shores” (LetterStream, 201).

Sometimes we make data fit the database because it is easier and cheaper to do so.

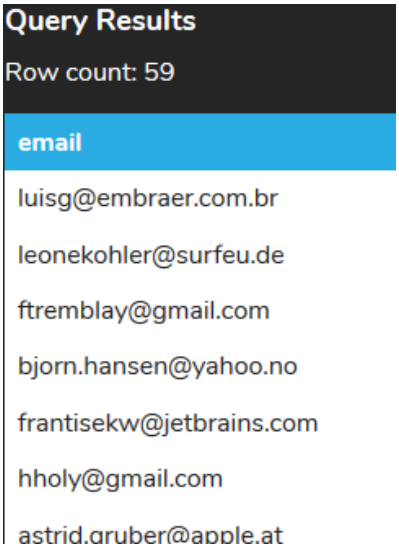
In this case, it is easier to abbreviate the city name than to increase the maximum size of the city attribute. It is also more efficient database design to do it that way; because we increased the maximum size, that attribute would occupy more storage space.

Suppose we want to use the customer table shown above to send out an email marketing campaign. Writing a query that pulled all the attributes would be wasteful. To select only email addresses from the customer table, we could write a SELECT statement that lists the column we want to display instead of using the asterisk *. Looking at the attribute list in the schema browser, as shown below, we see that there's an email attribute. That's probably the one.

Using the SELECT statement, we can alter the SELECT clause to display just the email addresses instead of all columns by replacing the * with the specific column name. Don't forget the ";" at the end of the command. The query statement would be changed to:

```
SELECT email
FROM customer;
```

Running this statement displays only email addresses:



TERM TO KNOW

Schema Browser

A list of table names, column names, and data types contained within a database.

2. Displaying Multiple Columns

Now let's suppose we also want the customers' first and last names so we can personalize the email messages. If we want to select more than one column, we need to separate out the column list using commas. For example:

```
SELECT first_name, last_name, email
FROM customer;
```

This would return the following result set:

Query Results

Row count: 59

first_name	last_name	email
Luís	Gonçalves	luisg@embraer.com.br
Leonie	Köhler	leonekohler@surfeu.de
François	Tremblay	ftremblay@gmail.com
Bjørn	Hansen	bjorn.hansen@yahoo.no
František	Wichterlová	frantisekw@ietbrains.com

Sometimes columns are not in the order in the table the way you want them displayed. If you want the column list to display in a different order, you can put the columns in the order you want using the SELECT statement:

```
SELECT email, last_name, first_name
FROM customer;
```

Query Results

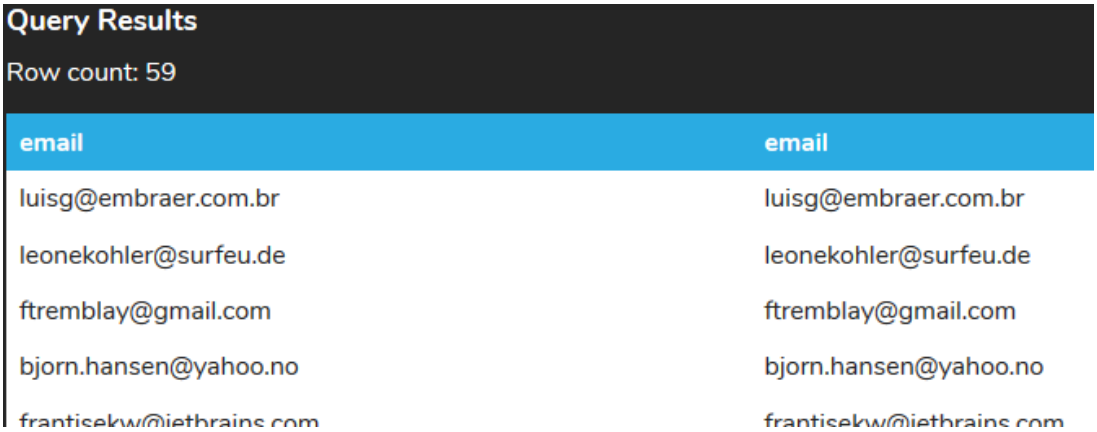
Row count: 59

email	last_name	first_name
luisg@embraer.com.br	Gonçalves	Luís
leonekohler@surfeu.de	Köhler	Leonie
ftremblay@gmail.com	Tremblay	François
bjorn.hansen@yahoo.no	Hansen	Bjørn
frantisekw@jetbrains.com	Wichterlová	František

You could even have the same column twice:

```
SELECT email, email
FROM customer;
```

This query returns:



Query Results

Row count: 59

email	email
luisg@embraer.com.br	luisg@embraer.com.br
leonekohler@surfeu.de	leonekohler@surfeu.de
ftremblay@gmail.com	ftremblay@gmail.com
bjorn.hansen@yahoo.no	bjorn.hansen@yahoo.no
frantisekw@jetbrains.com	frantisekw@jetbrains.com

With this capability, you can match the data output to your email merge system or format the data for automatic email generators. It will also come in handy when you start running queries that include calculations, which you will learn about later in this course.

WATCH

TRY IT

Your turn! Open the SQL tool by clicking on the LAUNCH DATABASE button below. Then, enter in one of the examples above and see how it works. Next, try your own choices for which columns you want the query to provide.

SUMMARY

In this lesson, you learned how to **use a SELECT statement to display data**, including how to **display multiple columns**. PostgreSQL's SELECT statement is a powerful and fundamental query command that retrieves data from one or more tables. You also learned that the data can be fetched from any table or tables specified by the columns you wish to retrieve. Finally, you learned that filtering the rows is also possible using various conditions. SELECT statements enable you to efficiently query and fetch data in PostgreSQL, allowing you to extract and manipulate information.

Source: THIS TUTORIAL WAS AUTHORED BY DR. VINCENT TRAN, PHD (2020) AND Faithe Wempen (2024) FOR SOPHIA LEARNING. PLEASE SEE OUR TERMS OF USE.

TERMS TO KNOW

Schema Browser

A list of table names, column names, and data types contained within a database.