AVG to Average Values

by Sophia

WHAT'S COVERED

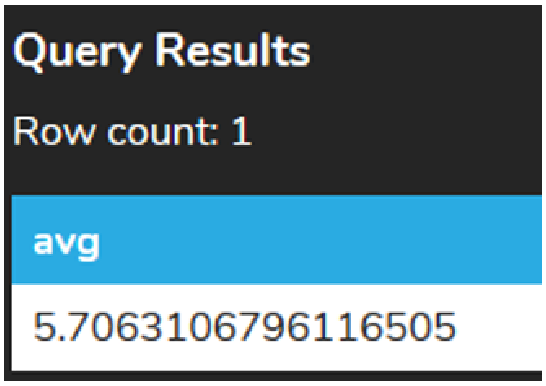
In this lesson, you will use the AVG function to average values of a given column within a table, in three parts. Specifically, this lesson will cover:

1. Introduction to the AVG Function

The **AVG function** (average) calculates the average of a set of values. This is one of the most commonly used aggregate functions. If we wanted to find the average total of all invoices from the invoice table, we could run something like this:

```
SELECT AVG(total)
FROM invoice;
```

This would return a value like:

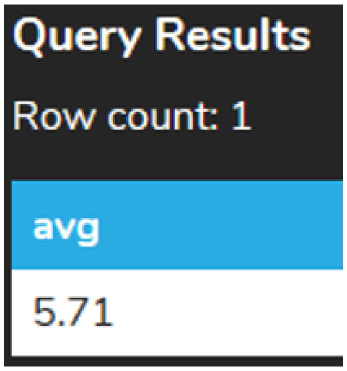


```
Query Results
Row count: 1
avg
5.7063106796116505
```

Since the value we're interested in is a price, it may make more sense to limit the result to two decimal places. You will learn more about formatting for decimal places in the next lesson. Unique to PostgreSQL, we can format the result as follows:

```
SELECT AVG(total)::numeric(10,2)
FROM invoice;
```

This will constrain the value in the total column to showing no more than 10 digits before the decimal point and no more than 2 digits after it.


Query Results

Row count: 1

avg
5.71

Other database systems use functions such as ROUND() to round the value or TRUNC() to truncate the value to a certain position without rounding. You will learn more about TRUNC in the next section of this lesson, and you will learn about ROUND in the next lesson.

TERM TO KNOW

AVG Function

A function that calculates the average value of the entries in a numeric column. It is commonly used in statistical calculations and data analysis.

2. TRUNCATE (TRUNC) Function

Numeric values or timestamps can be truncated using the **TRUNC() function** in PostgreSQL. This function can truncate numeric values and can reset the timestamp precision to a specified unit.

The TRUNC function takes the following parameters:

- numeric_expression: The numeric value or column to be truncated.
- decimal_places: The number of decimal places to which the value should be truncated. If omitted, the TRUNC function removes all decimal places and returns the integer portion of the value.

If the decimal_places parameter is positive for numeric values, the TRUNC function truncates the numeric_expression to that number of decimal places towards zero. If the parameter is negative, the function truncates the value to the left of the decimal point. If omitted or set to 0, the TRUNC function removes all decimal places.

For timestamp values, the TRUNC function truncates the timestamp to the specified unit. For example, if the unit is “month,” it truncates the timestamp to the beginning of the month, setting the day, hour, minute, and second values to zero.

For the following example, we have a number with five digits following the decimal point, and we want to truncate it to two decimal places. The first argument in the function is the original number, and the second one is the desired number of decimal places. The output in this case will be 3.14.

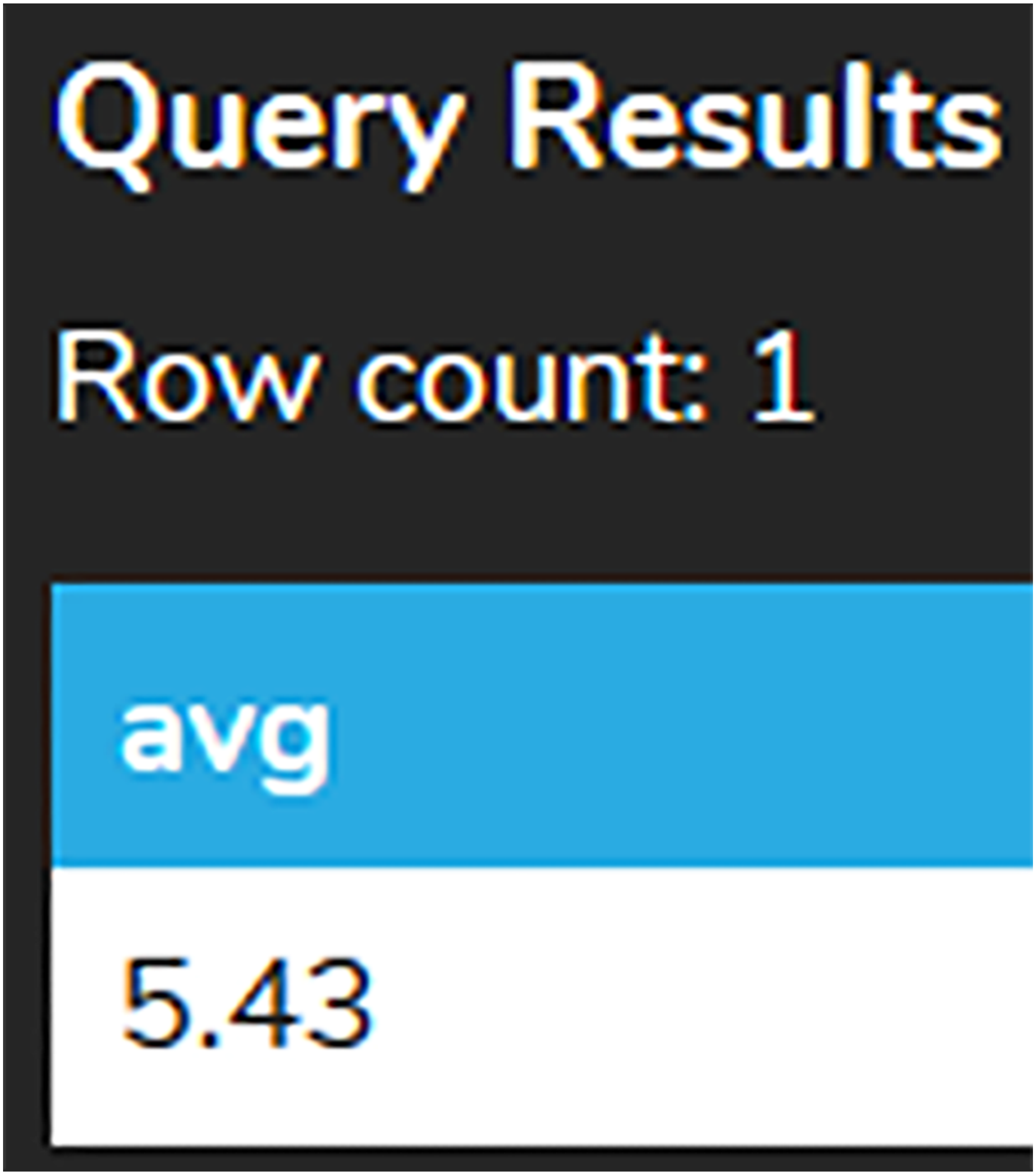
```
SELECT TRUNC(3.14159, 2);
```

For this example, we have a standard Date Time format, and we want to truncate it to HOUR and not show minutes and seconds. We get 15:00:00 for our answer.

```
SELECT TRUNC('2023-06-23 15:30:45'::timestamp, 'hour');
```

The TRUNC function is not needed with the AVG function because as you learned earlier in this lesson, the AVG function can be configured to limit itself to a certain number of decimal places, like this:

```
SELECT AVG(total)::numeric(10,2)
FROM invoice
WHERE customer_id = 2;
```



```
Query Results
Row count: 1
avg
5.43
```

TERM TO KNOW

TRUNC() Function

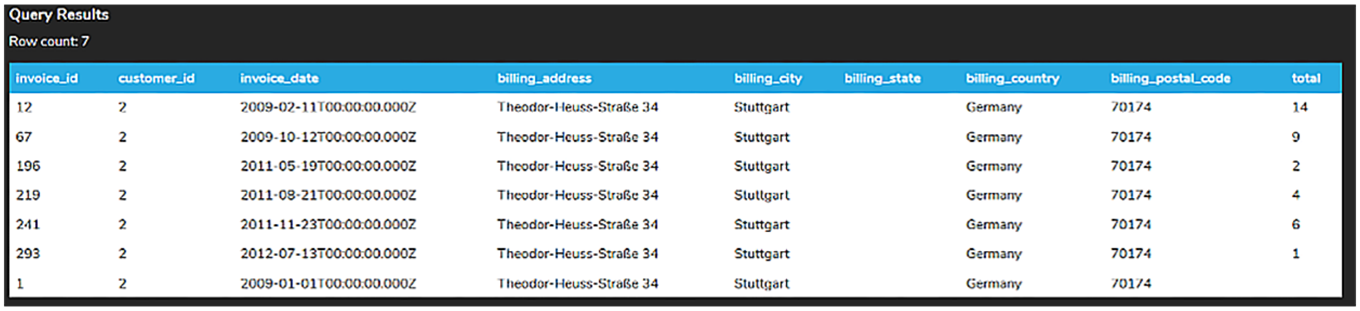
The TRUNC function in PostgreSQL truncates a numeric value to a specified number of decimal places or removes the decimal part altogether.

3. Dealing With 0 and NULL

It's important to note that the AVG function operates only on non-null values in columns with a numeric data type. It will not work for text (VARCHAR) or (CHAR), even if they happen to hold data consisting only of numeric digits. For example, consider what happens if we set one of the seven invoices to have the total as NULL:

```
UPDATE invoice
SET total = NULL
WHERE invoice_id = 1;
```

If we queried for the customer_id = 2, we would see that the total has no value. This is different from having a value of 0.



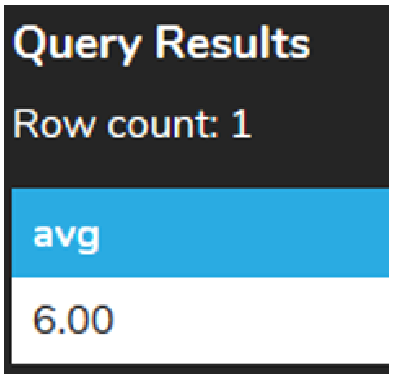
Query Results

Row count: 7

invoice_id	customer_id	invoice_date	billing_address	billing_city	billing_state	billing_country	billing_postal_code	total
12	2	2009-02-11T00:00:00.000Z	Theodor-Heuss-Straße 34	Stuttgart		Germany	70174	14
67	2	2009-10-12T00:00:00.000Z	Theodor-Heuss-Straße 34	Stuttgart		Germany	70174	9
196	2	2011-05-19T00:00:00.000Z	Theodor-Heuss-Straße 34	Stuttgart		Germany	70174	2
219	2	2011-08-21T00:00:00.000Z	Theodor-Heuss-Straße 34	Stuttgart		Germany	70174	4
241	2	2011-11-23T00:00:00.000Z	Theodor-Heuss-Straße 34	Stuttgart		Germany	70174	6
293	2	2012-07-13T00:00:00.000Z	Theodor-Heuss-Straße 34	Stuttgart		Germany	70174	1
1	2	2009-01-01T00:00:00.000Z	Theodor-Heuss-Straße 34	Stuttgart		Germany	70174	

Calculating the average would result in the following because we don't count NULL items in an average:

```
SELECT AVG(total)::numeric(10,2)
FROM invoice
WHERE customer_id = 2;
```



Query Results

Row count: 1

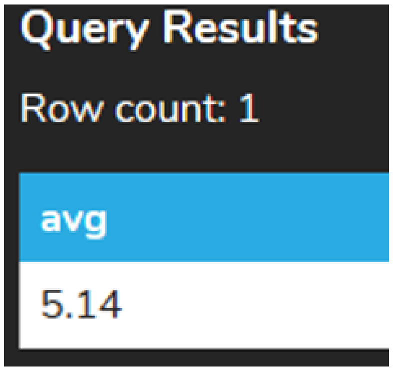
avg
6.00

However, if we updated that same invoice to set the total to 0, like this, it would return the following:

```
UPDATE invoice
SET total = 0
WHERE invoice_id = 1;
```

Running the same calculation, it returns this:

```
SELECT AVG(total)::numeric(10,2)
FROM invoice
WHERE customer_id = 2;
```

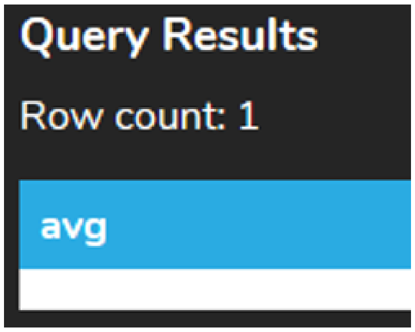


```
Query Results
Row count: 1
avg
5.14
```

In the first case, even though there are seven rows, the row with the null value does not count. The query runs the following calculation: (14 + 9 + 2 + 4 + 6 + 1)/6 = 6.00. In the second case, since the 0 is added instead of the null value, it counts it as part of the calculation: (14 + 9 + 2 + 4 + 6 + 1 + 0)/7 = 5.14.

Similar to the SUM function, if we only have NULL values, using the AVG would return NULL.

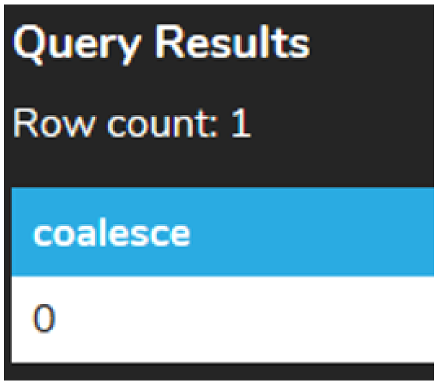
```
SELECT AVG(total)
FROM invoice
WHERE invoice_id < 1;
```



```
Query Results
Row count: 1
avg
```

To have a 0 returned, you would need to use the COALESCE() function, covered in the previous lesson, that would return the second argument if the first argument was NULL.

```
SELECT COALESCE(AVG(total),0)
FROM invoice
WHERE invoice_id < 1;
```


Query Results

Row count: 1

coalesce
0

WATCH

TRY IT

Your turn! Open the SQL tool by clicking on the LAUNCH DATABASE button below. Then, enter 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 that in PostgreSQL, the **AVG function** calculates the average of a set of numeric values. Taking a column or an expression as input, it returns the average value of the selected elements. The AVG function calculates a value by summing all values and dividing them by the count. You also learned that the AVG function is commonly used in statistical calculations and data analysis to calculate the average value of a dataset. The AVG function accepts data from any numeric data type, including integers, floating-point numbers, and decimals. You can use it to determine the average value of a column or a set of values in PostgreSQL.

You learned that by using the **TRUNC function** in PostgreSQL, numeric values can be truncated to a specific number of decimal places, or the decimal part removed entirely. With the TRUNC function, you can specify a precision argument and the number to be truncated. The precision value indicates how many decimal places should be truncated, effectively removing digits beyond that point. TRUNC removes the decimal part entirely when precision is not specified or if precision is **0 or NULL**. PostgreSQL's TRUNC function is useful for adjusting numeric values, such as removing decimal places or rounding down values.

TERMS TO KNOW

AVG Function

A function that calculates the average value of the entries in a numeric column. It is commonly used in statistical calculations and data analysis.

TRUNC() Function

A function in PostgreSQL that truncates a numeric value to a specified number of decimal places or removes the decimal part altogether.