HAVING to Filter On Aggregates

by Sophia

WHAT'S COVERED

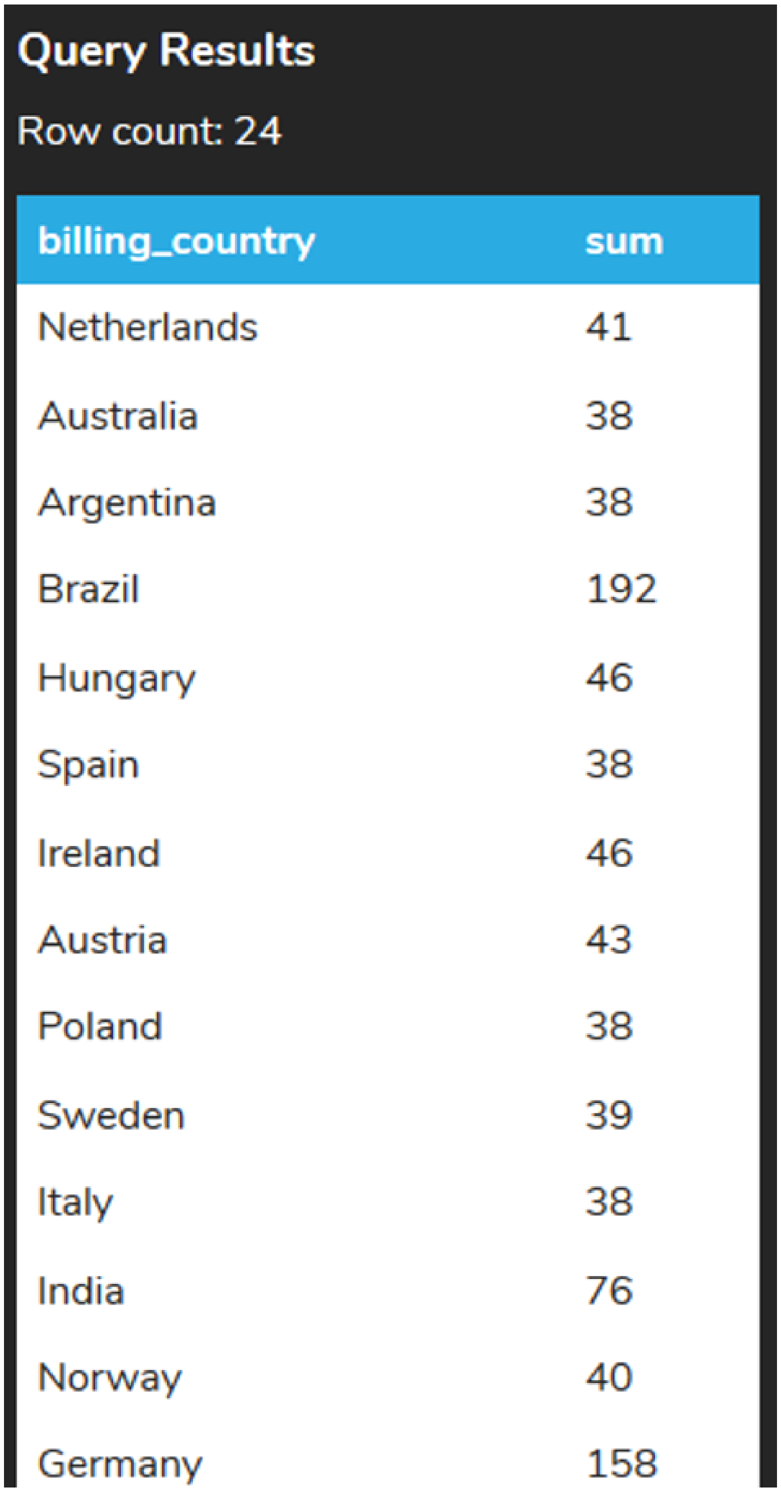
In this lesson, you will use the HAVING clause to place a filter condition on groups and aggregates, in two parts. Specifically, this lesson will cover:

1. Using the HAVING Clause

Earlier in the course, you learned to use the WHERE clause in queries to filter records. However, WHERE is not applicable to aggregate functions, so instead we use the **HAVING clause** with them. The HAVING clause filters groups of rows based on a set of conditions. It enables us to specify a search condition for a group or an aggregate.

Let's take a look at the invoice table to find the SUM of the total that is grouped by country:

```
SELECT billing_country, SUM(total)
FROM invoice
GROUP BY billing_country;
```



Query Results

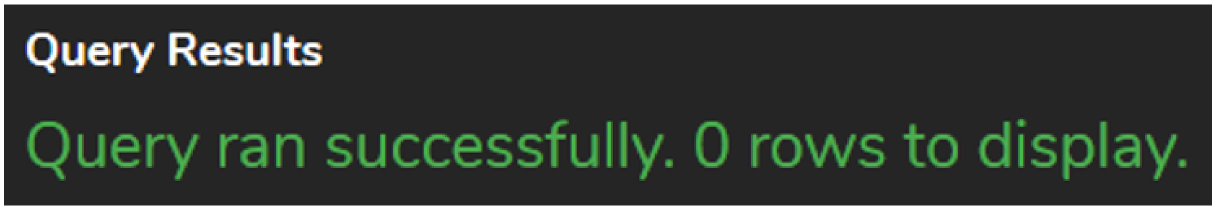
Row count: 24

billing_country	sum
Netherlands	41
Australia	38
Argentina	38
Brazil	192
Hungary	46
Spain	38
Ireland	46
Austria	43
Poland	38
Sweden	39
Italy	38
India	76
Norway	40
Germany	158

Suppose we want to list only the countries where the sum of the total column is greater than 50. You might think that adding WHERE to this query would do the trick, as in the following example. This query will run, but it does not return any data.

```
SELECT billing_country, SUM(total)
FROM invoice
```

```
WHERE total > 50
GROUP BY billing_country;
```

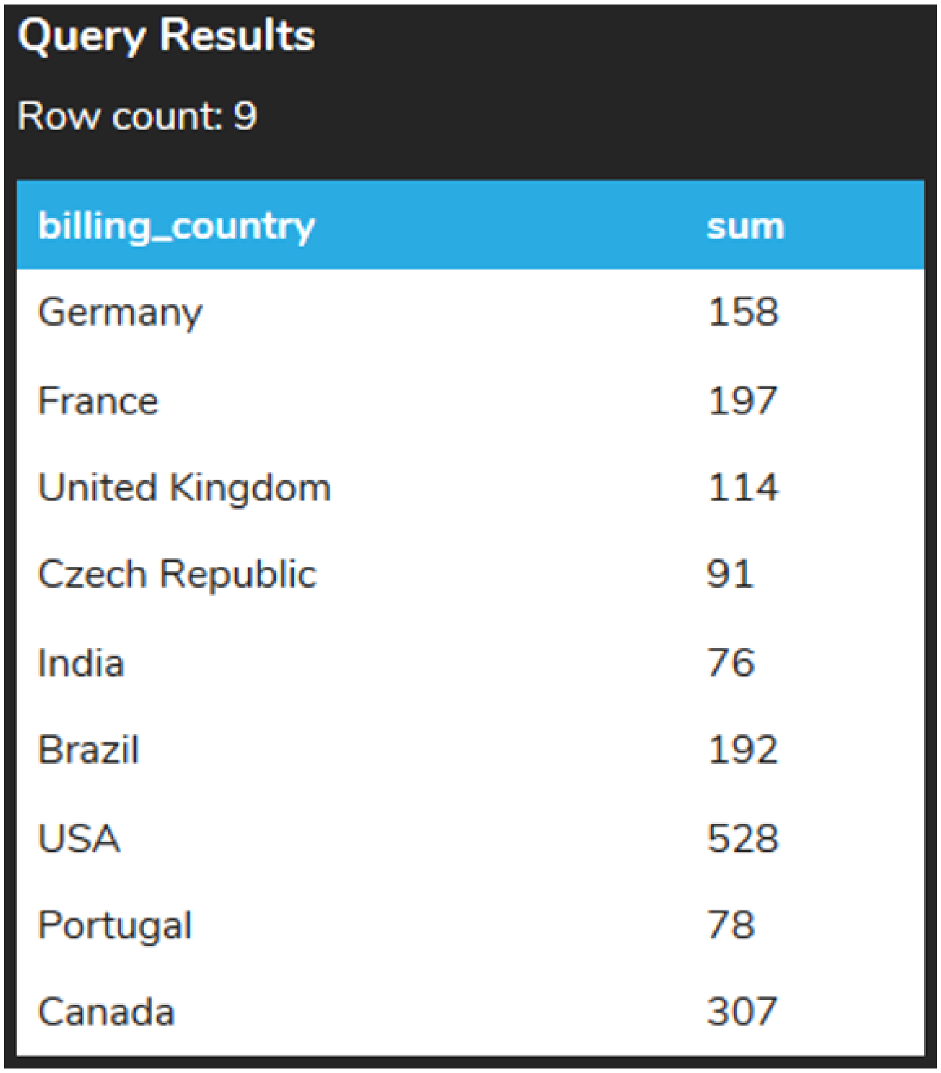


Query Results

Query ran successfully. 0 rows to display.

Instead, we could use the HAVING clause, which is designed to work cooperatively with GROUP BY. Here's what that query would look like:

```
SELECT billing_country, SUM(total)
FROM invoice
GROUP BY billing_country
HAVING SUM(total) > 50;
```



Query Results

Row count: 9

billing_country	sum
Germany	158
France	197
United Kingdom	114
Czech Republic	91
India	76
Brazil	192
USA	528
Portugal	78
Canada	307

TERM TO KNOW

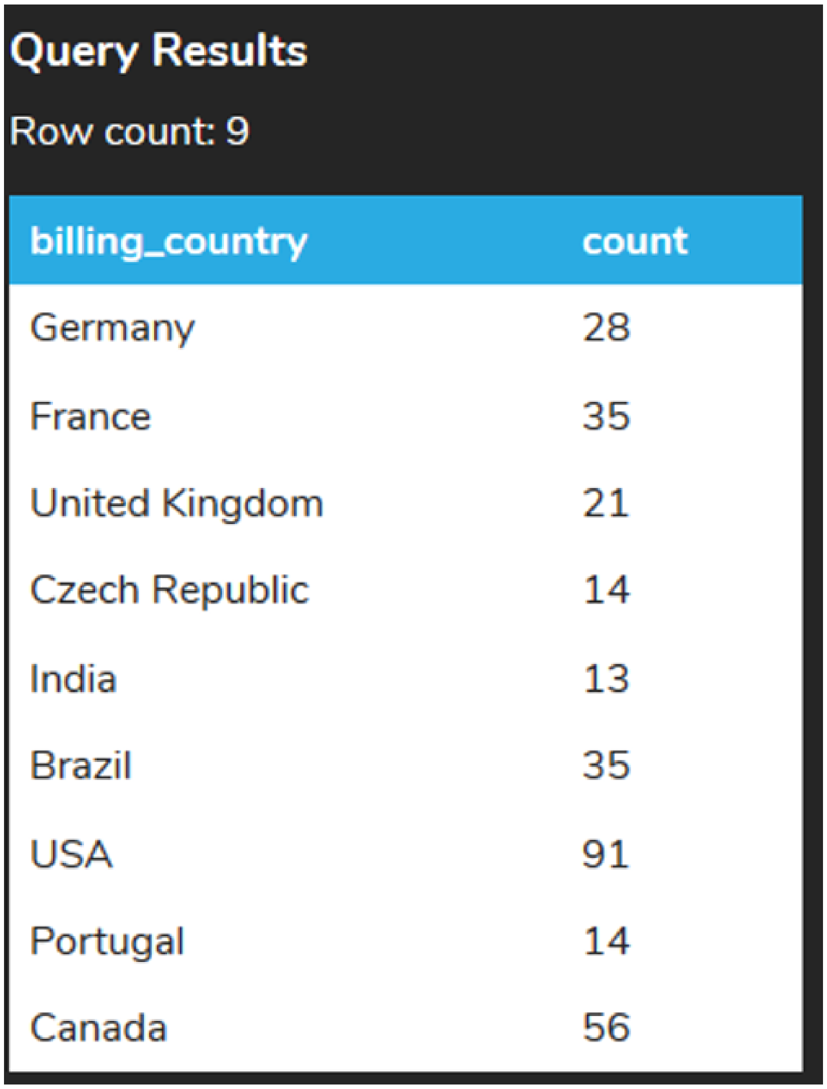
HAVING Clause

A clause that enables you to filter the results of a GROUP BY clause according to specified conditions.

2. More Aggregate Uses

The HAVING clause can be used with a variety of SELECT statements that aggregate and group data. For example, if we wanted to show the number of invoices having the total amount > 50 by country, we can change the SUM to COUNT in the SELECT clause:

```
SELECT billing_country, COUNT(total)
FROM invoice
GROUP BY billing_country
HAVING SUM(total) > 50;
```



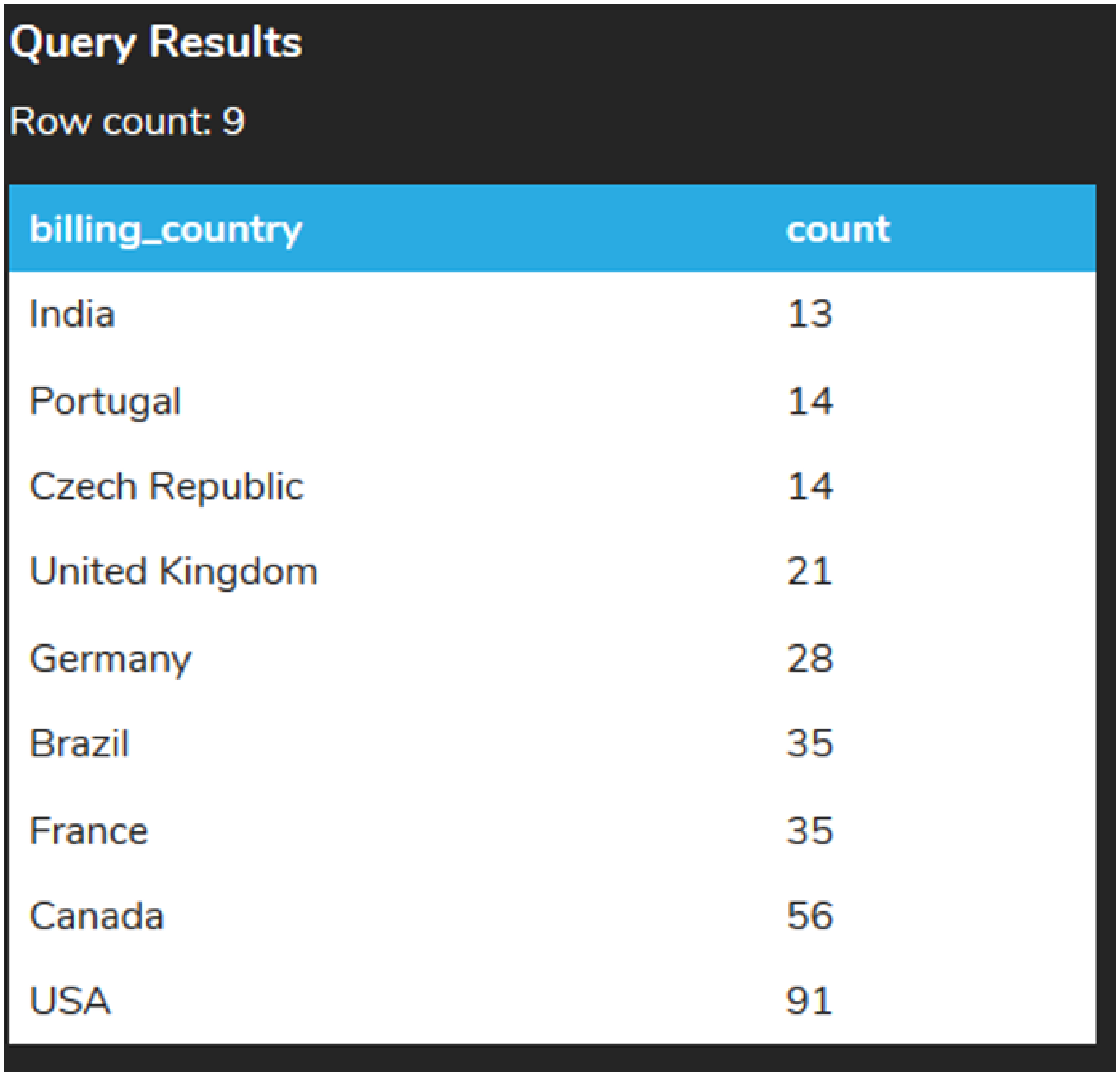
Query Results

Row count: 9

billing_country	count
Germany	28
France	35
United Kingdom	21
Czech Republic	14
India	13
Brazil	35
USA	91
Portugal	14
Canada	56

We could also sort the results using the aggregate function ORDER BY SUM() as well:

```
SELECT billing_country, COUNT(total)
FROM invoice
GROUP BY billing_country
HAVING SUM(total) > 50
ORDER BY SUM(total);
```



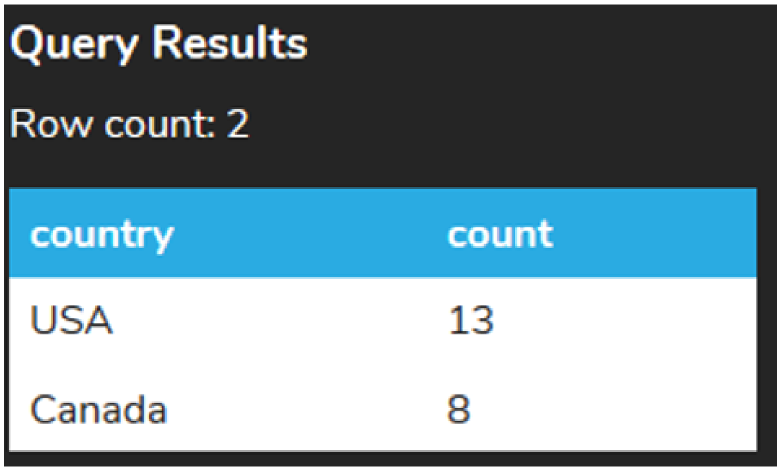
Query Results

Row count: 9

billing_country	count
India	13
Portugal	14
Czech Republic	14
United Kingdom	21
Germany	28
Brazil	35
France	35
Canada	56
USA	91

We can use a variety of aggregate functions like the COUNT function to help filter. For example, we may want to find the countries and the number of customers that have a count greater than five:

```
SELECT country, COUNT(*)
FROM customer
GROUP BY country
HAVING COUNT(*) > 5;
```

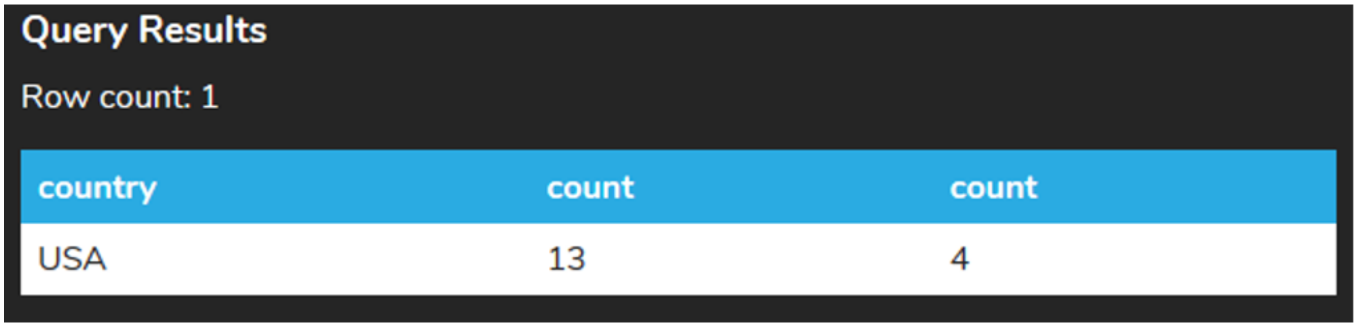

Query Results

Row count: 2

country	count
USA	13
Canada	8

We could have multiple conditions to filter based on aggregate values by using the AND and OR operators in the HAVING clause. We can filter the group further to ensure that the number of customers also checks for those with a fax count greater than two.

```
SELECT country, COUNT(*), COUNT(fax)
FROM customer
GROUP BY country
HAVING COUNT(*) > 5 AND COUNT(fax) > 2;
```


Query Results

Row count: 1

country	count	count
USA	13	4

Counting a specific column counts only the non-null values, so in the example of COUNT(fax), if a record did not have anything in the fax column, that row would not be counted. In contrast, COUNT(*) counts all rows.

As you learned earlier in the course, if null values are causing querying problems in a table, you can use COALESCE and fill in those null columns.

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 that in PostgreSQL, the GROUP BY clause groups rows based on one or more columns. You also learned that you can partition the data using the GROUP BY clause based on unique combinations of values in the specified columns. Using **HAVING clauses** together with GROUP BY enables the grouped data to be filtered based on aggregated values based on conditions. By specifying conditions, you can filter out groups that do not meet the specified criteria. Finally, you learned that PostgreSQL's GROUP BY and HAVING clauses enable you to analyze complex data and apply conditions to more **aggregated uses**.

TERMS TO KNOW

HAVING Clause

A clause that enables you to filter the results of a GROUP BY clause according to specified conditions.