Section 4 Grouping Data

Introduction to PostgreSQL GROUP BY clause

The GROUP BY clause divides the rows returned from the SELECT statement into groups. For each group, you can apply an aggregate function e.g., SUM () to calculate the sum of items or COUNT () to get the number of items in the groups.

The following statement illustrates the basic syntax of the GROUP BY clause:

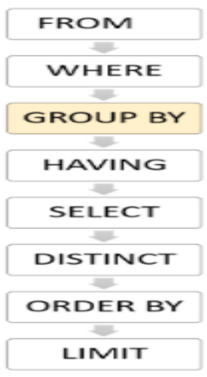
```
SELECT
   column_1,
   column_2,
   ...,
   aggregate_function(column_3)
FROM
   table_name
GROUP BY
   column_1,
   column_2,
   ...;
```

In this syntax:

- First, select the columns that you want to group e.g., column1 and column2, and column that you want to apply an aggregate function (column3).
- Second, list the columns that you want to group in the GROUP BY clause.

The statement clause divides the rows by the values of the columns specified in the GROUP BY clause and calculates a value for each group.

It’s possible to use other clauses of the SELECT statement with the GROUP BY clause.

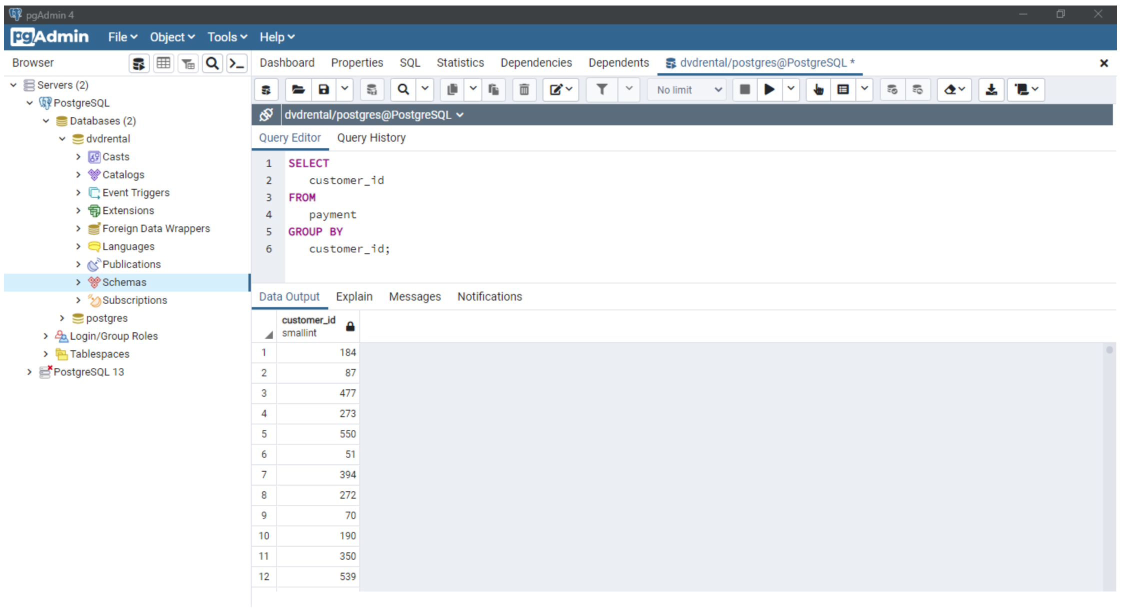


PostgreSQL evaluates the GROUP BY clause after the FROM and WHERE clauses and before the HAVING SELECT, DISTINCT, ORDER BY and LIMIT clauses.

PostgreSQL GROUP BY clause

1) Using PostgreSQL GROUP BY without an aggregate function example

You can use the GROUP BY clause without applying an aggregate function. The following query gets data from the payment table and groups the result by customer id.



```
SELECT
    customer_id
FROM
    payment
GROUP BY
    customer_id;
```

```
    customer_id
    smallint
1          184
2           87
3          477
4          273
5          550
6           51
7          394
8          272
9           70
10         190
11         350
12         539
```

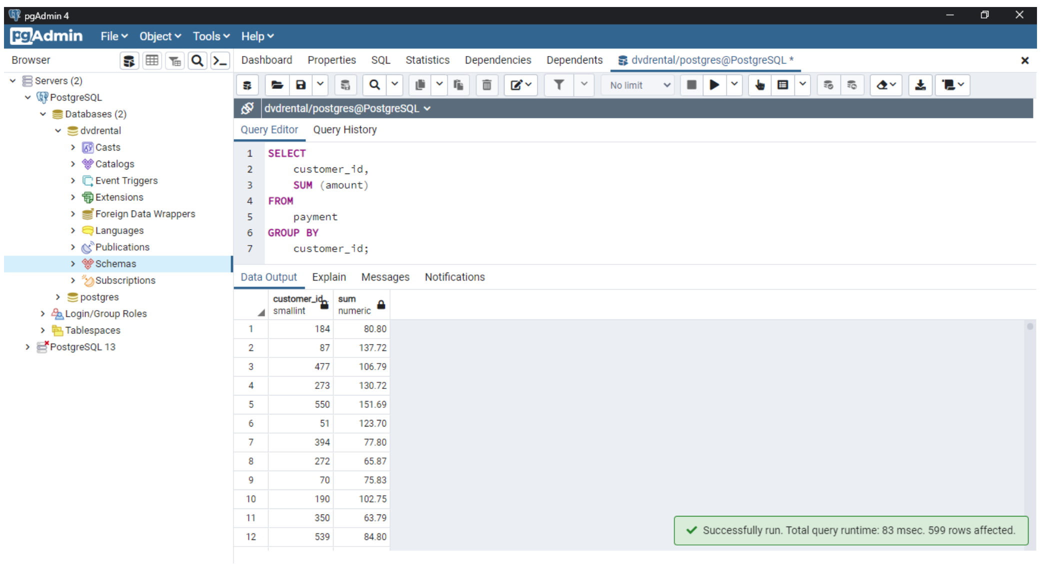
In this case, the GROUP BY works like the DISTINCT clause that removes duplicate rows from the result set.

2) Using PostgreSQL GROUP BY with SUM () function example

The GROUP BY clause is useful when it is used in conjunction with an aggregate function.

For example, to select the total amount that each customer has been paid, you use the GROUP BY clause to divide the rows in the payment table into groups grouped by customer id. For each group, you calculate the total amounts using the SUM () function.

The following query uses the GROUP BY clause to get total amount that each customer has been paid:



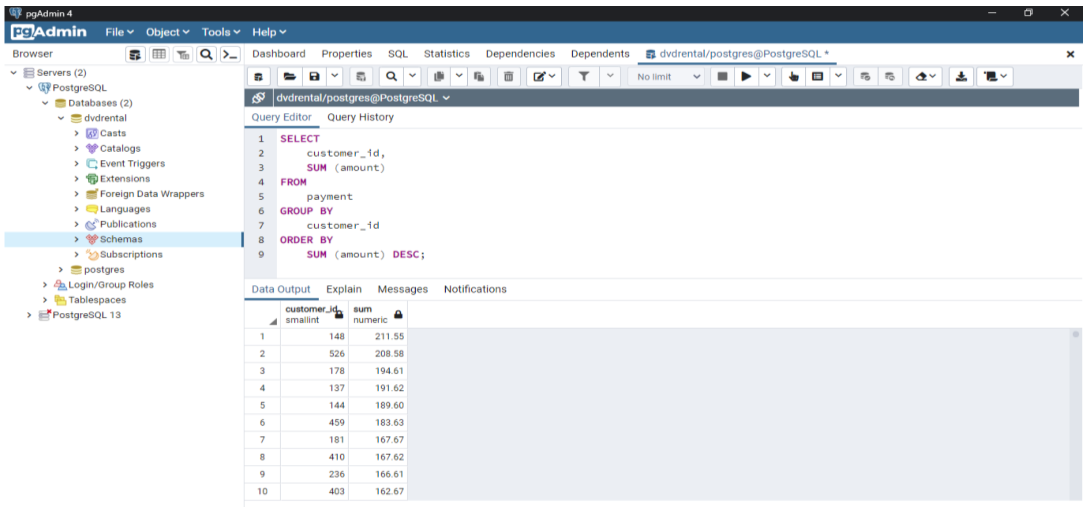
```
SELECT
    customer_id,
    SUM (amount)
FROM
    payment
GROUP BY
    customer_id;
```

```
    customer_id  sum
    smallint     numeric
1          184    80.80
2           87   137.72
3          477   106.79
4          273   130.72
5          550   151.69
6           51   123.70
7          394    77.80
8          272    65.87
9           70    75.83
10         190   102.75
11         350    63.79
12         539    84.80
```

```
Successfully run. Total query runtime: 83 msec. 599 rows affected.
```

The GROUP BY clause sorts the result set by customer id and adds up the amount that belongs to the same customer. Whenever the customer_id changes, it adds the row to the returned result set.

The following statement uses the ORDER BY clause with GROUP BY clause to sort the groups:



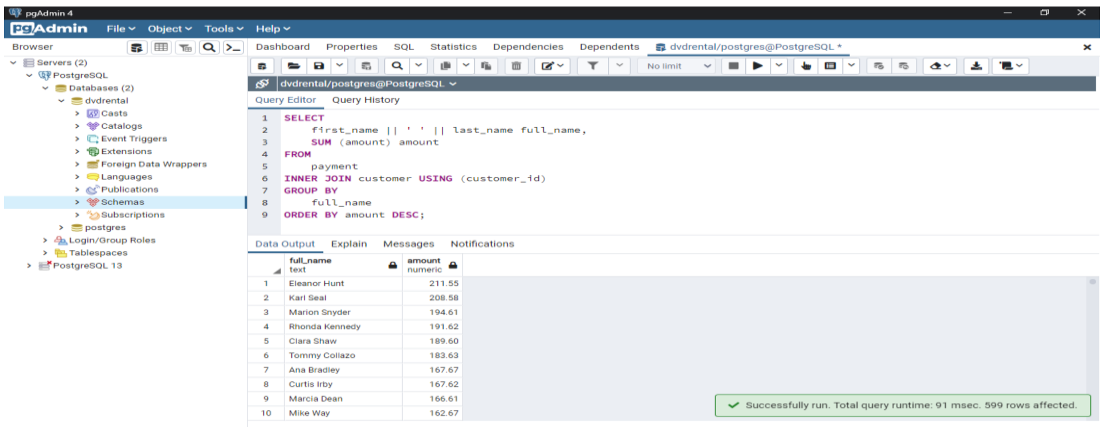
```
SELECT
    customer_id,
    SUM (amount)
FROM
    payment
GROUP BY
    customer_id
ORDER BY
    SUM (amount) DESC;
```

```
    customer_id  sum
    smallint     numeric
1   148          211.55
2   526          208.58
3   178          194.61
4   137          191.62
5   144          189.60
6   459          183.63
7   181          167.67
8   410          167.62
9   236          166.61
10  403          162.67
```

3) Using PostgreSQL GROUP BY clause with the JOIN clause

The following statement uses the GROUP BY clause with the INNER JOIN clause the get the total amount paid by each customer.

Unlike the previous example, this query joins the payment table with the customer table and group customers by their names.



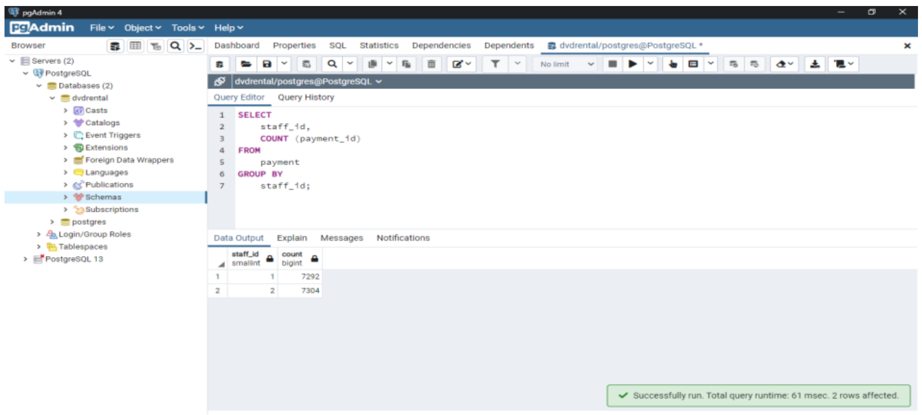
```
SELECT
    first_name || ' ' || last_name full_name,
    SUM (amount) amount
FROM
    payment
INNER JOIN customer USING (customer_id)
GROUP BY
    full_name
ORDER BY amount DESC;
```

```
    full_name        amount
    text             numeric
1   Eleanor Hunt     211.55
2   Karl Seal        208.58
3   Marion Snyder    194.61
4   Rhonda Kennedy   191.62
5   Clara Shaw       189.60
6   Tommy Collazo    183.63
7   Ana Bradley      167.67
8   Curtis Irby      167.62
9   Marcia Dean      166.61
10  Mike Way         162.67
```

```
Successfully run. Total query runtime: 91 msec. 599 rows affected.
```

4) Using PostgreSQL GROUP BY with COUNT () function example

To find the number of payment transactions that each staff has been processed, you group the rows in the payment table by the values in the staff_id column and use the COUNT () function to get the number of transactions:



```
SELECT
    staff_id,
    COUNT (payment_id)
FROM
    payment
GROUP BY
    staff_id;
```

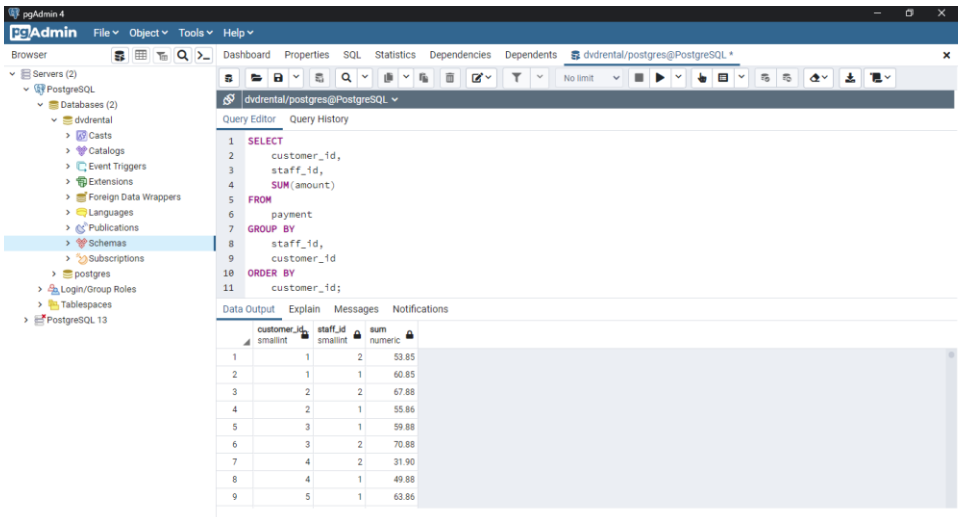
```
    staff_id  count
    smallint  bigint
1   1         7292
2   2         7304
```

```
Successfully run. Total query runtime: 61 msec. 2 rows affected.
```

The GROUP BY clause divides the rows in the payment into groups and groups them by value in the staff_id column. For each group, it returns the number of rows by using the COUNT () function.

5) Using PostgreSQL GROUP BY with multiple columns

The following example uses multiple columns in the GROUP BY clause:



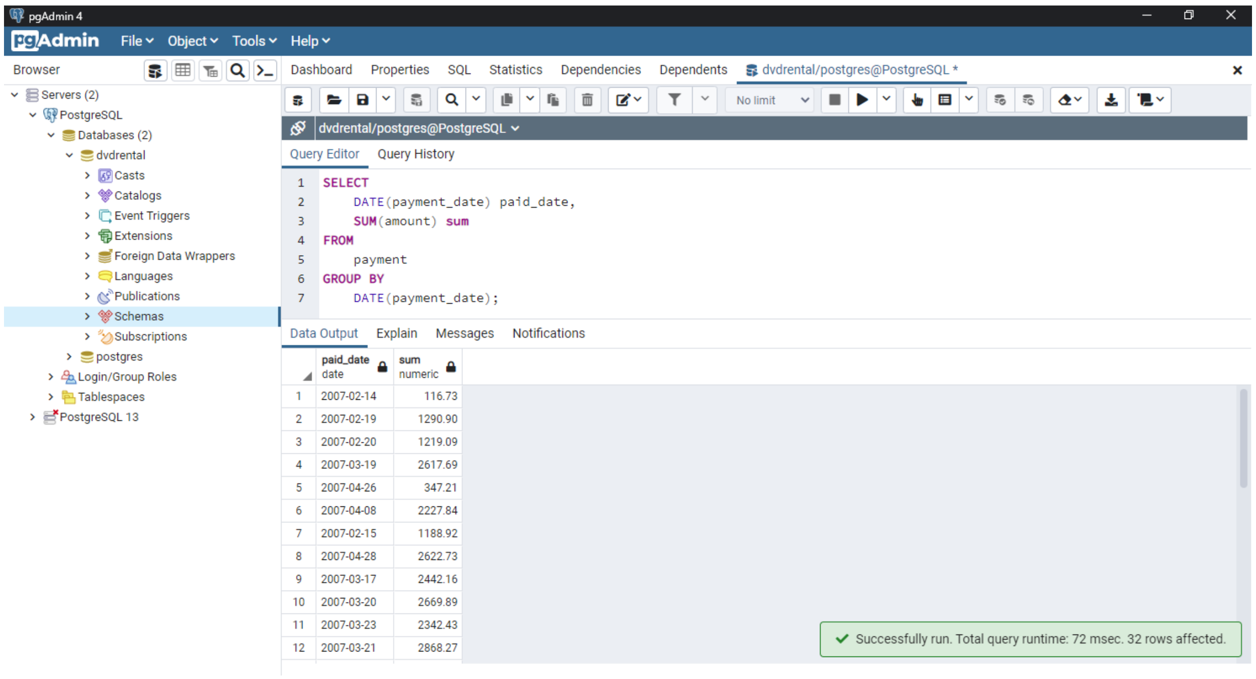
```sql
SELECT
    customer_id,
    staff_id,
    SUM(amount)
FROM
    payment
GROUP BY
    staff_id,
    customer_id
ORDER BY
    customer_id;
```

	customer_id smallint	staff_id smallint	sum numeric
1	1	2	53.85
2	1	1	60.85
3	2	2	67.88
4	2	1	55.86
5	3	1	59.88
6	3	2	70.88
7	4	2	31.90
8	4	1	49.88
9	5	1	63.86

In this example, the GROUP BY clause divides the rows in the payment table by the values in the customer_id and staff_id columns. For each group of (customer_id, staff_id), the SUM () calculates the total amount.

6) Using PostgreSQL GROUP BY clause with date column

The payment_date is a timestamp column. To group payments by dates, you use the DATE () function to convert timestamps to dates first and then group payments by the result date:



```sql
SELECT
    DATE(payment_date) paid_date,
    SUM(amount) sum
FROM
    payment
GROUP BY
    DATE(payment_date);
```

	paid_date date	sum numeric
1	2007-02-14	116.73
2	2007-02-19	1290.90
3	2007-02-20	1219.09
4	2007-03-19	2617.69
5	2007-04-26	347.21
6	2007-04-08	2227.84
7	2007-02-15	1188.92
8	2007-04-28	2622.73
9	2007-03-17	2442.16
10	2007-03-20	2669.89
11	2007-03-23	2342.43
12	2007-03-21	2868.27

Successfully run. Total query runtime: 72 msec. 32 rows affected.

Introduction to PostgreSQL HAVING clause

The HAVING clause specifies a search condition for a group or an aggregate. The HAVING clause is often used with the GROUP BY clause to filter groups or aggregates based on a specified condition.

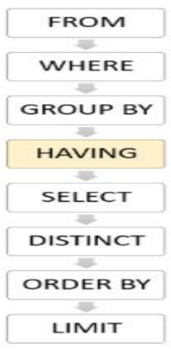
The following statement illustrates the basic syntax of the HAVING clause:

```
SELECT
        column1,
        aggregate_function (column2)
FROM
        table_name
GROUP BY
        column1
HAVING
        condition;
```

In this syntax, the group by clause returns rows grouped by the column1. The HAVING clause specifies a condition to filter the groups.

It's possible to add other clauses of the SELECT statement such as JOIN, LIMIT, and FETCH etc.

PostgreSQL evaluates the HAVING clause after the FROM, WHERE, GROUP BY, and before the SELECT, DISTINCT, ORDER BY and LIMIT clauses.



Since the HAVING clause is evaluated before the SELECT clause, you cannot use column aliases in the HAVING clause. Because at the time of evaluating the HAVING clause, the column aliases specified in the SELECT clause are not available.

HAVING vs. WHERE

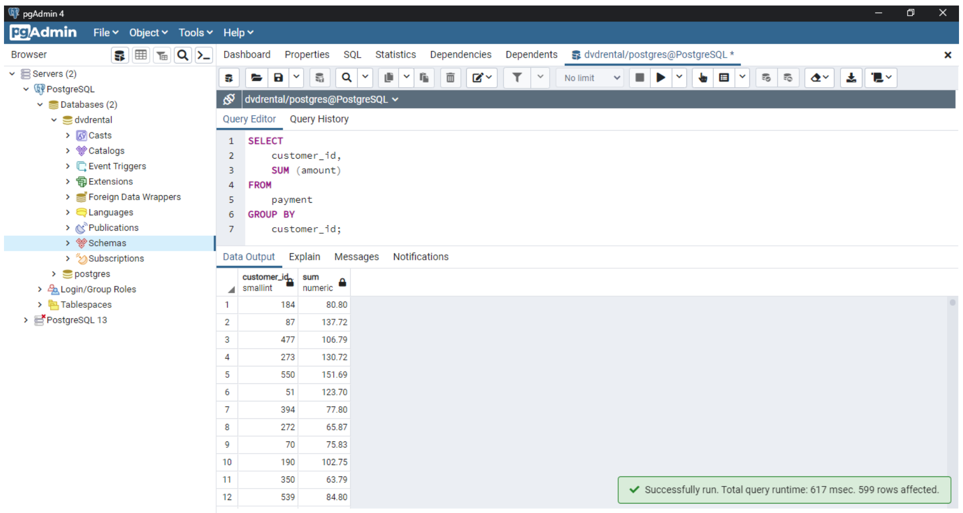
The WHERE clause allows you to filter rows based on a specified condition. However, the HAVING clause allows you to filter groups of rows according to a specified condition.

In other words, the WHERE clause is applied to rows while the HAVING clause is applied to groups of rows.

PostgreSQL HAVING clause examples

1) Using PostgreSQL HAVING clause with SUM function example

The following query uses the GROUP BY clause with the SUM () function to find the total amount of each customer:

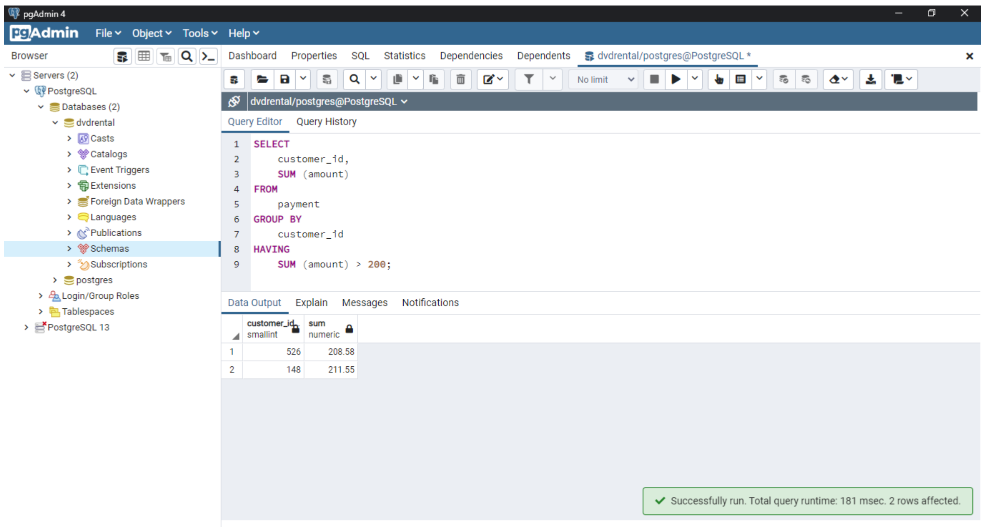


```
SELECT
    customer_id,
    SUM (amount)
FROM
    payment
GROUP BY
    customer_id;
```

```
   customer_id  sum
   smallint     numeric
1          184    80.80
2           87   137.72
3          477   106.79
4          273   130.72
5          550   151.69
6           51   123.70
7          394    77.80
8          272    65.87
9           70    75.83
10         190   102.75
11         350    63.79
12         539    84.80
```

```
Successfully run. Total query runtime: 617 msec. 599 rows affected.
```

The following statement adds the HAVING clause to select the only customers who have been spending more than 200:



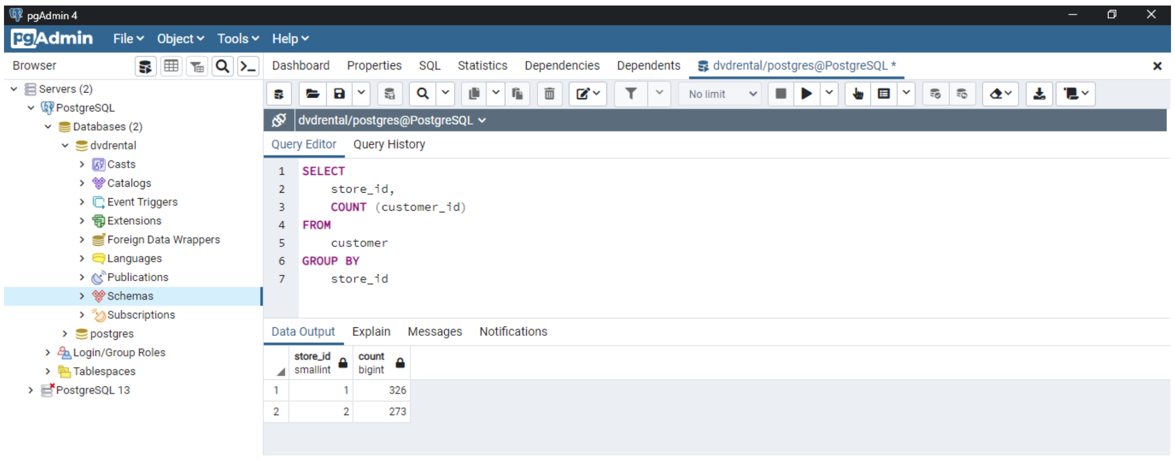
```
SELECT
    customer_id,
    SUM (amount)
FROM
    payment
GROUP BY
    customer_id
HAVING
    SUM (amount) > 200;
```

```
   customer_id  sum
   smallint     numeric
1          526   208.58
2          148   211.55
```

```
Successfully run. Total query runtime: 181 msec. 2 rows affected.
```

2) PostgreSQL HAVING clause with COUNT example

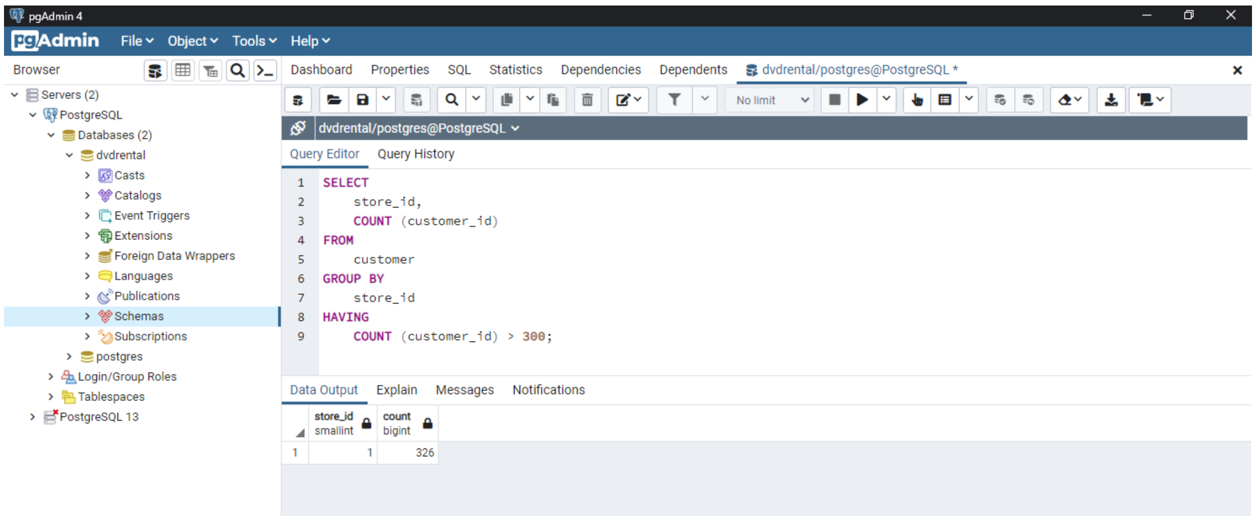
The following query uses the GROUP BY clause to find the number of customers per store:



```
SELECT
    store_id,
    COUNT (customer_id)
FROM
    customer
GROUP BY
    store_id
```

```
   store_id  count
   smallint  bigint
1         1    326
2         2    273
```

The following statement adds the HAVING clause to select the store that has more than 300 customers:



```
SELECT
    store_id,
    COUNT (customer_id)
FROM
    customer
GROUP BY
    store_id
HAVING
    COUNT (customer_id) > 300;
```

	store_id smallint	count bigint
1	1	326