In this lesson we are looking at Functions. There are numbers of functions that we can use those are built in SQL and some those are unique to Postgres. We can also create our own functions which is quite interesting and something we could not de before in relational databases.

Let’s look at some functions that we have already seen before. The first function that we have seen before is called the COUNT function. It is used to count number of occurrences of things. We will use the COUNT function to count the number of purchases some of our customers have made.

SELECT customers.first\_name, customers.last\_name, COUNT(purchases.id) AS purchase\_count

FROM customers

INNER JOIN purchases ON customers.id = purchases.customer\_id

GROUP BY customers.id;

|  |  |  |
| --- | --- | --- |
| first\_name  character varying (100) | last\_name  character varying (255) | purchase\_count  bigint |
| Craig | Scott | 2 |
| Michael | Adam | 4 |
| Rolf | Smith | 2 |

One of the other SQL functions are the AVG function.

SELECT \* FROM items;

|  |  |  |
| --- | --- | --- |
| Name  Character varying (100) | Id  Integer | Price  Numeric (10,2) |
| Fountain Pen | 2 | 11.30 |
| Screen | 5 | 275.50 |
| Hard Drive | 6 | 89.99 |
| Pen | 1 | 5.00 |
| Ink | 3 | 5.00 |
| Laptop | 7 | 1500.00 |
| Laptop | 8 | 15.00 |

SELECT AVG(items.price) FROM items;

|  |
| --- |
| avg  numeric |
| 218.708181818182 |

This is the average price of all the items’ prices added together divided by the number of items.

We can also calculate the revenue generated per order.

SELECT \* FROM purchases;

|  |  |  |
| --- | --- | --- |
| **id**  **integer** | **item\_id**  **integer** | **customer\_id**  **integer** |
| 1 | 4 | 1 |
| 2 | 5 | 1 |
| 3 | 6 | 1 |
| 4 | 1 | 3 |
| 5 | 3 | 5 |
| 6 | 2 | 5 |
| 7 | 4 | 2 |

SELECT AVG(items.price) FROM items

INNER JOIN purchases ON items.id = purchases.item\_id;

|  |
| --- |
| avg  numeric |
| 72.2587500000000000 |

This gives us the average price of all the items that we sold unlike the last query which gave us the average price of all the items that we have in our store. That’s the difference there between these two queries.

We can also calculate the sale that has generated most revenue in our business and we already know how to do that using ORDER BY together with LIMIT.

SELECT items.name, items.price FROM items

INNER JOIN purchases ON items.id = purchases.item\_id

ORDER BY items.price DESC;

|  |  |
| --- | --- |
| name  character varying (255) | price  numeric (10,2) |
| Screen | 275.50 |
| Hard Drive | 89.99 |
| Hard Drive | 89.99 |
| Hard Drive | 89.99 |
| Fountain Pen | 11.30 |
| Fountain Pen | 11.30 |
| Ink | 5.00 |
| Ink | 5.00 |

SELECT items.name, items.price FROM items

INNER JOIN purchases ON items.id = purchases.item\_id

ORDER BY items.price DESC

LIMIT 1;

|  |  |
| --- | --- |
| name  character varying (255) | price  numeric (10,2) |
| Screen | 275.50 |

This is one way of doing it and it is a pretty long way that to do it.

To make lives easy we have a function called MAX that sorts out the maximum value from a column.

SELECT MAX(items.price) FROM items

INNER JOIN purchases ON items.id = purchases.id;

|  |
| --- |
| max  numeric |
| 275.50 |

This gives us the item that brought us the maximum revenue, but we lose the item name.

Now we will look at a very useful construct called HAVING. The HAVING construct allows us to filter after the data collection from our table has taken place.

SELECT customers.first\_name, customers.last\_name, COUNT (purchases.id) AS purchase\_count

FROM customers

INNER JOIN purchases ON customers.id = purchases.customer\_id

GROUP BY customer.id;

|  |  |  |
| --- | --- | --- |
| first\_name  character varying (100) | last\_name  character varying (255) | purchase\_count  bigint |
| Craig | Scott | 2 |
| Michael | Adam | 4 |
| Rolf | Smith | 2 |

This query gives us the first names and the last names of our customers and how many purchases they have done.

Now we want to get the customers that have over three purchases.

There is no way we can do that with all the stuffs that we have learned so far.

There is a SQL command which we have not learned about and it is called HAVING. Using this command, we can figure out the amount of purchases some customers made above a certain number.

SELECT customers.first\_name, customers.last\_name, COUNT(purchases.id) AS purchase\_count

FROM customers

INNER JOIN purchases ON customers.id = purchases.customer\_id

GROUP BY customers.id

HAVING COUNT(purchases.id) > 3;

This HAVING happens after we GROUP BY customers.id