In this lesson we are looking at VIEWs. VIEWs are not something we have seen before.

A VIEW is essentially a result of a query. Its not quite that simple. We can do things with VIEWs that we cannot do with basic results. Think of a VIEW as a camera looking at a table and then we can interact with that camera. We can interact with that VIEW, in anyway we can interact with a normal table.

Let’s see how we can create a VIEW and different ways we can interact with it.

For example, we want to get total revenue per customer, essentially how much they have spent in our business. Let’s first do that with a normal query.

SELECT \* FROM customer

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| first\_name  character varying (100) | id  integer | last\_name  character varying (255) | id  integer | item\_id  integer | customer\_id  integer |
| Rolf | 1 | Smith | 2 | 5 | 1 |
| Rolf | 1 | Smith | 3 | 6 | 1 |
| Michael | 5 | Adam | 5 | 3 | 5 |
| Michael | 5 | Adam | 6 | 2 | 5 |
| Craig | 4 | Scott | 8 | 2 | 4 |
| Craig | 4 | Scott | 9 | 3 | 4 |
| Michael | 5 | Adam | 10 | 6 | 5 |

SELECT \* FROM customer

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

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

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| first\_name  character varying (100) | id  integer | last\_name  character varying (255) | id  integer | item\_id  integer | customer\_id  integer | name  character varying (255) | id  integer | price  numeric |
| Rolf | 1 | Smith | 2 | 5 | 1 | Screen | 5 | 275.50 |
| Rolf | 1 | Smith | 3 | 6 | 1 | Hard Drive | 6 | 89.99 |
| Michael | 5 | Adam | 5 | 3 | 5 | Ink | 3 | 5.00 |
| Michael | 5 | Adam | 6 | 2 | 5 | Fountain Pen | 2 | 11.30 |
| Craig | 4 | Scott | 8 | 2 | 4 | Fountain Pen | 2 | 11.30 |
| Craig | 4 | Scott | 9 | 3 | 4 | Ink | 3 | 5.00 |
| Michael | 5 | Adam | 10 | 6 | 5 | Hard Drive | 6 | 89.99 |

Now we want to calculate the total that each customer has spent in our business.

SELECT customers.first\_name, customers.last\_name, SUM(items.price) FROM customer

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

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

GROUP BY customers.id;

|  |  |  |
| --- | --- | --- |
| first\_name  character varying (100) | last\_name  character varying (255) | sum  numeric |
| Michael | Adam | 106.29 |
| Rolf | Smith | 365.49 |
| Craig | Scott | 16.30 |

Now, this maybe quite a useful result for a business, to know which customers are the top customers and to send them promotions and things like that.

The thing is it is great way to see who our top customers are, but it is not a user-friendly process to write this much queries every time to display the top customers in our business.

That is why there is way to save a query as a VIEW. Let’s do this creating a view.

CREATE VIEW total\_revenue\_per\_customer AS

SELECT customers.first\_name, customers.last\_name, SUM(items.price) FROM customer

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

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

GROUP BY customers.id;

This is the natural syntax to create a VIEW, it starts with CREATE VIEW and then followed by the name of the view that the user sets, and after that we write AS followed by the query that we are saving as the view.

If we are the query, Postgres tells us that our query is successful. Now we will select the VIEW to see what got saved in our VIEW.

SELECT \* FROM total\_revenue\_per\_customer;

|  |  |  |
| --- | --- | --- |
| first\_name  character varying (100) | last\_name  character varying (255) | sum  numeric |
| Michael | Adam | 106.29 |
| Rolf | Smith | 365.49 |
| Craig | Scott | 16.30 |

So, we get our result from the VIEW that we saved, without typing the full query for it. Notice that we do not have the customer id because we did not save it in our VIEW.

If we want, we can DROP the VIEW as below,

DROP VIEW total\_revenue\_per\_customer;

And our VIEW dropped and then we can rewrite the query to create the VIEW again but, in this case, we will include the *customers.id*,

CREATE VIEW total\_revenue\_per\_customer AS

SELECT customers.id, customers.first\_name, customers.last\_name, SUM(items.price) FROM customer

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

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

GROUP BY customers.id;

That is how we DROP a VIEW and we can recreate it. This query works and we can select our VIEW again.

SELECT \* FROM total\_revenue\_per\_customer;

|  |  |  |  |
| --- | --- | --- | --- |
| id  integer | first\_name  character varying (100) | last\_name  character varying (255) | sum  numeric |
| 5 | Michael | Adam | 106.29 |
| 1 | Rolf | Smith | 365.49 |
| 4 | Craig | Scott | 16.30 |

The reason we added the id here so that we can show you what happens when we add a new purchase for one of these customers.

Let’s see our items’ list again,

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 |

So, we plan to add item with id 6, as the item purchased by one of our customers.

We need to add that first in our purchases’ table.

SELECT \* FROM purchases;

|  |  |  |
| --- | --- | --- |
| id  integer | item\_id  integer | customer\_id  integer |
| 2 | 5 | 1 |
| 3 | 6 | 1 |
| 5 | 3 | 5 |
| 6 | 2 | 5 |
| 8 | 2 | 4 |
| 9 | 3 | 4 |
| 10 | 6 | 5 |

Now we will INSERT the item to this purchases’ table.

INSERT INTO purchases

VALUES (11, 6, 5);

After we run the query, Postgres tells us it ran successfully.

Now we will see check our VIEW to see the increase in money spent amount that should have increased for the customer with id 5.

SELECT \* FROM total\_revenue\_per\_customer;

|  |  |  |  |
| --- | --- | --- | --- |
| id  integer | first\_name  character varying (100) | last\_name  character varying (255) | sum  numeric |
| 5 | Michael | Adam | 196.28 |
| 1 | Rolf | Smith | 365.49 |
| 4 | Craig | Scott | 16.30 |

As we can see the customer with id number 6, Michael Adam, his sum has increased.

This means that the VIEW is updatable. Whenever we are selecting our VIEW, at the background the query that we have saved is being executed.

VIEW is not saving the result, rather it is saving the query that we are running.

Let’s now say that we want to the view the customers that has spent over $150.

SELECT \* FROM total\_revenue\_per\_customer WHERE sum > 150;

|  |  |  |  |
| --- | --- | --- | --- |
| id  integer | first\_name  character varying (100) | last\_name  character varying (255) | sum  numeric |
| 5 | Michael | Adam | 196.28 |
| 1 | Rolf | Smith | 365.49 |

CRAETE VIEW awesome\_customers AS

SELECT \* FROM total\_revenue\_per\_customer WHERE sum > 150;

SELECT \* FROM awesome\_customers;

|  |  |  |  |
| --- | --- | --- | --- |
| id  integer | first\_name  character varying (100) | last\_name  character varying (255) | sum  numeric |
| 5 | Michael | Adam | 196.28 |
| 1 | Rolf | Smith | 365.49 |

SELECT \* FROM awesome\_customers ORDER BY sum DESC;

|  |  |  |  |
| --- | --- | --- | --- |
| id  integer | first\_name  character varying (100) | last\_name  character varying (255) | sum  numeric |
| 1 | Rolf | Smith | 365.49 |
| 5 | Michael | Adam | 196.28 |

Inserting in a VIEW.

CREATE VIEW expensive\_item AS

SELECT \* FROM items WHERE price > 100;

SELECT \* FROM expensive\_items;

|  |  |  |
| --- | --- | --- |
| name  character varying (100) | id  integer | price  numeric |
| Screen | 5 | 275.50 |
| Laptop | 7 | 1500.00 |

The VIEW provides us with all the items those are above $100.

Now we will INSERT into this VIEW.

INSERT INTO expensive\_items (id, name, price)

VALUES (9, ‘DSLR’, 400.00);

SELECT \* FROM expensive\_items;

|  |  |  |
| --- | --- | --- |
| name  character varying (255) | id  integer | price  numeric (10,2) |
| Screen | 5 | 275.50 |
| Laptop | 7 | 1500.00 |
| DSLR | 9 | 400.00 |

INSERT INTO expensive\_items(id, name, price)

VALUES (10, ‘Pencil’, 2.00);

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 |
| DSLR | 9 | 400.00 |
| Pencil | 10 | 2.00 |

DROP VIEW expensive\_items;

CREATE VIEW expensive\_items AS

SELECT \* FROM items WHERE price > 100

WITH LOCAL CHECK OPTION;

After we execute the above query, LOCAL CHECK OPTION restricts a user to enter items below $100 to the VIEW.

INSER INTO expensive\_items(id, name, price)

VALUES (11, ‘Pencil’, 2.00);

Trying to run this query, Postgres gives an ERROR, telling us “new row violates check option for view ‘expensive\_items’”.

It does not give us much information about the error itself, but it does not let us enter the pencil in there.

It would let us enter values above $100, and if someone tries to enter $100 with the CHECK POINT restriction in place, it will block the user from entering $100.

If we want to enter items with price of $100.

Then we would do as following,

DROP VIEW expensive\_items;

CREATE VIEW expensive\_items AS

SELECT \* FROM items WHERE price >= 100

WITH LOCAL CHECK OPTION;

Our query runs successfully. Now we will insert a pencil worth of $100.

INSERT INTO expensive\_items(id, name, price)

VALUES (11, ‘Pencil’, 100.00);

Our query runs successfully, and the pencil is entered in the VIEW expensive\_items.