# VIEWS

A view is a database object that is of a stored query.The fields in a view are fields from one or more real tables in the database.. In other words, a PostgreSQL view is a logical table that represents data of one or more underlying tables through a SELECT statement. Notice that a view does not store data physically except for a materialized view.

In simple terms, view is a virtual table based on the result set of an sql query.

A view can be very useful in some cases such as:

- A view helps simplify the complexity of a query because you can query a view, which is based on a complex query, using a simple SELECT statement.
- Like a table, you can grant permission to users through a view that contains specific data that the users are authorized to see.
- A view provides a consistent layer even if the columns of the underlying table change.

## Creating Postgresql Views

To create a view, we use the CREATE VIEW statement. The simplest syntax of the CREATE VIEW statement is as follows:

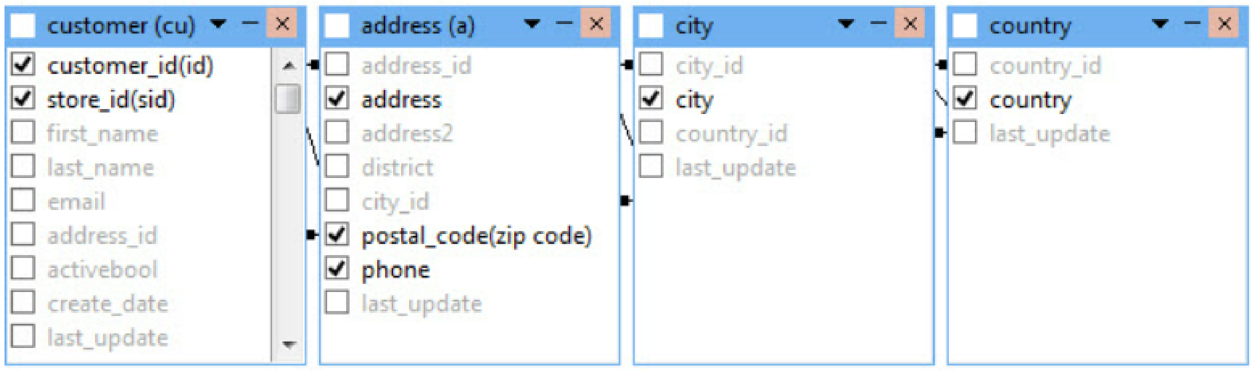
```
CREATE VIEW view_name AS query;
```

First, you specify the name of the view after the CREATE VIEW clause, then you put a query after the AS keyword. A query can be a simple SELECT statement or a complex SELECT statement with joins.

### PostgreSQL CREATE VIEW example

For example, in our sample database, we have four tables:

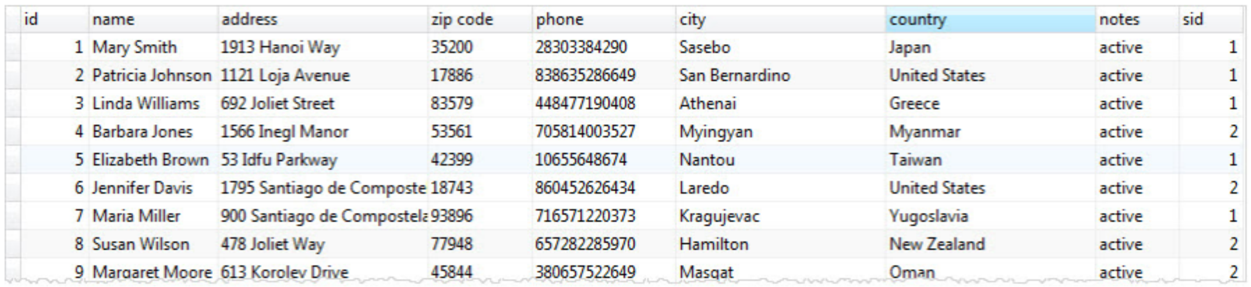
1. customer – stores all customer data
2. address – stores address of customers
3. city – stores city data
4. country– stores country data



Normally if you want to get a complete customers data, you normally construct a join statement as follows:

```
SELECT cu.customer_id AS id,
   cu.first_name || ' ' || cu.last_name AS name,
   a.address,
   a.postal_code AS "zip code",
   a.phone,
   city.city,
   country.country,
       CASE
           WHEN cu.activebool THEN 'active'
           ELSE ''
       END AS notes,
   cu.store_id AS sid
  FROM customer cu
    INNER JOIN address a USING (address_id)
    INNER JOIN city USING (city_id)
    INNER JOIN country USING (country_id);
```

The result of the query is as shown in the screenshot below:



| id | name | address | zip code | phone | city | country | notes | sid |
|---|---|---|---|---|---|---|---|---|
| 1 | Mary Smith | 1913 Hanoi Way | 35200 | 28303384290 | Sasebo | Japan | active | 1 |
| 2 | Patricia Johnson | 1121 Loja Avenue | 17886 | 838635286649 | San Bernardino | United States | active | 1 |
| 3 | Linda Williams | 692 Joliet Street | 83579 | 448477190408 | Athenai | Greece | active | 1 |
| 4 | Barbara Jones | 1566 Inegl Manor | 53561 | 705814003527 | Myingyan | Myanmar | active | 2 |
| 5 | Elizabeth Brown | 53 Idfu Parkway | 42399 | 10655648674 | Nantou | Taiwan | active | 1 |
| 6 | Jennifer Davis | 1795 Santiago de Composte | 18743 | 860452626434 | Laredo | United States | active | 2 |
| 7 | Maria Miller | 900 Santiago de Compostel | 93896 | 716571220373 | Kragujevac | Yugoslavia | active | 1 |
| 8 | Susan Wilson | 478 Joliet Way | 77948 | 657282285970 | Hamilton | New Zealand | active | 2 |
| 9 | Margaret Moore | 613 Korolev Drive | 45844 | 380657522649 | Masqat | Oman | active | 2 |

This query is quite complex. However, you can create a view named customer_master as follows:

```
CREATE VIEW customer_master AS
  SELECT cu.customer_id AS id,
    cu.first_name || ' ' || cu.last_name AS name,
    a.address,
    a.postal_code AS "zip code",
    a.phone,
    city.city,
    country.country,
        CASE
            WHEN cu.activebool THEN 'active'
            ELSE ''
        END AS notes,
    cu.store_id AS sid
   FROM customer cu
     INNER JOIN address a USING (address_id)
     INNER JOIN city USING (city_id)
     INNER JOIN country USING (country_id);
```

From now on, whenever you need to get a complete customer data, you just query it from the view by executing the following simple SELECT statement:

```
SELECT
        *
FROM
        customer_master;
```

This query produces the same result as the complex one with joins above.

### Example 2

The following SQL creates a view that selects every product in the "Products" table with a price higher than the average price:

```
CREATE VIEW [Products Above Average Price] AS
SELECT ProductName, Price
FROM Products
WHERE Price > (SELECT AVG(Price) FROM Products);
```

We can query the view above as follows:

```
SELECT * FROM [Products Above Average Price];
```

## Changing PostgreSQL Views

To change the defining query of a view, you use the CREATE VIEW statement with OR REPLACE addition as follows:

```
CREATE OR REPLACE view_name
AS
query
```

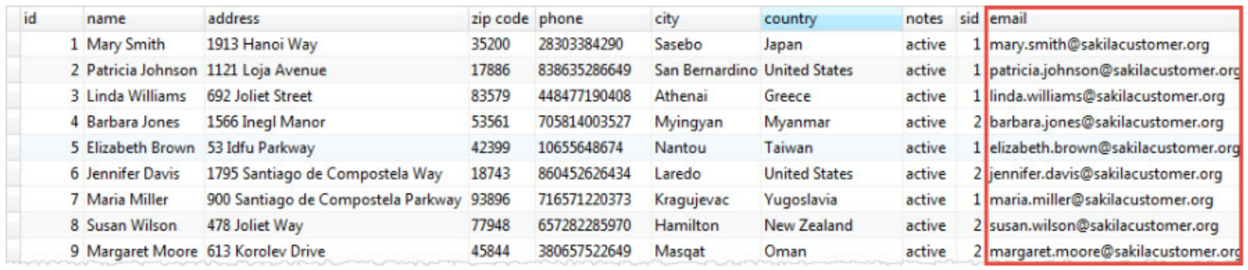
PostgreSQL does not support removing an existing column in the view, at least up to version 9.4. If you try to do it, you will get an error message: “[Err] ERROR:  cannot drop columns from view”. The query must generate the same columns that were generated when the view was created. To be more specific, the new columns must have the same names, same data types, and in the same order as they were created. However, PostgreSQL allows you to append additional columns at the end of the column list.

For example, you can add an email to the customer_master  view as follows:

```
CREATE VIEW customer_master AS
  SELECT cu.customer_id AS id,
    cu.first_name || ' ' || cu.last_name AS name,
    a.address,
    a.postal_code AS "zip code",
    a.phone,
    city.city,
    country.country,
        CASE
            WHEN cu.activebool THEN 'active'
            ELSE ''
        END AS notes,
    cu.store_id AS sid,
    cu.email
   FROM customer cu
     INNER JOIN address a USING (address_id)
     INNER JOIN city USING (city_id)
     INNER JOIN country USING (country_id);
```

Now, if you select data from the customer_master view, you will see the email column at the end of the list.

```
SELECT
        *
FROM
        customer_master;
```



| id | name | address | zip code | phone | city | country | notes | sid | email |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Mary Smith | 1913 Hanoi Way | 35200 | 28303384290 | Sasebo | Japan | active | 1 | mary.smith@sakilacustomer.org |
| 2 | Patricia Johnson | 1121 Loja Avenue | 17886 | 838635286649 | San Bernardino | United States | active | 1 | patricia.johnson@sakilacustomer.org |
| 3 | Linda Williams | 692 Joliet Street | 83579 | 448477190408 | Athenai | Greece | active | 1 | linda.williams@sakilacustomer.org |
| 4 | Barbara Jones | 1566 Inegl Manor | 53561 | 705814003527 | Myingyan | Myanmar | active | 2 | barbara.jones@sakilacustomer.org |
| 5 | Elizabeth Brown | 53 Idfu Parkway | 42399 | 10655648674 | Nantou | Taiwan | active | 1 | elizabeth.brown@sakilacustomer.org |
| 6 | Jennifer Davis | 1795 Santiago de Compostela Way | 18743 | 860452626434 | Laredo | United States | active | 2 | jennifer.davis@sakilacustomer.org |
| 7 | Maria Miller | 900 Santiago de Compostela Parkway | 93896 | 716571220373 | Kragujevac | Yugoslavia | active | 1 | maria.miller@sakilacustomer.org |
| 8 | Susan Wilson | 478 Joliet Way | 77948 | 657282285970 | Hamilton | New Zealand | active | 2 | susan.wilson@sakilacustomer.org |
| 9 | Margaret Moore | 613 Korolev Drive | 45844 | 380657522649 | Masqat | Oman | active | 2 | margaret.moore@sakilacustomer.org |

To change the definition of a view, you use the ALTER VIEW statement. For example, you can change the name of the view from customer_master to customer_info by using the following statement:

```
ALTER VIEW customer_master RENAME TO customer_info;
```

PostgreSQL allows you to set a default value for a column name, change the view's schema, set or reset options of a view. For detailed information on the altering view's definition, check out the PostgreSQL ALTER VIEW statement.

## Removing Postgresql Views

To remove an existing view in PostgreSQL, you use DROP VIEW statement as follows:

```
DROP VIEW [ IF EXISTS ] view_name;
```

You specify the name of the view that you want to remove after the DROP VIEW clause. Removing a view that does not exist in the database will result in an error. To avoid this, you normally add the IF EXISTS option to the statement to instruct PostgreSQL to remove the view if it exists, otherwise, do nothing.

For example, to remove the customer_info view that you have created, you execute the following query:

```
DROP VIEW IF EXISTS customer_info;
```

The view customer_infois removed from the database.

In this tutorial, we have shown you how to create, alter, and remove PostgreSQL views.

### Views for Complex Queries

Suppose A and B are two tables and we wan't to select data from both of the tables. For that, we have to use SQL JOINS.

However using the JOIN each time could be a tedious task. For that, we can create a view to fetch records easily.

Let's create a view,

```
CREATE VIEW order_details AS
SELECT Customers.customer_id, Customers.first_name, Orders.amount
FROM Customers
JOIN Orders
ON Customers.customer_id = Orders.customer_id;
```

Now, to select the data, we can run

```
SELECT *
FROM order_details;
```

Here, the SQL command selects data from the view order_details.