Advanced POSTGRESQL

1>CREATE FUNCTION **3>TRIGGER**
2>ALIAS **4>INDEX**

POSTGRESQL "**Create function**":-

Syntax:-

```
CREATE FUNCTION function_name ([parameter_list])
RETURNS return_type AS $$
-- Function body
BEGIN
  -- Statements
END;
$$ LANGUAGE language_name;

SELECT function_name();
```

Example:-

```
CREATE OR REPLACE FUNCTION factorial(n INTEGER)
RETURNS INTEGER AS
$$
BEGIN
 IF n = 0 THEN
   RETURN 1;
 ELSE
   RETURN n * factorial(n - 1);
 END IF;
END;
$$ LANGUAGE plpgsql;

Select factorial(7);
```

Example 2:-

```
CREATE OR REPLACE FUNCTION get_car_price(latest_model INT, old_model INT)
RETURNS INTEGER AS $$
DECLARE
  difference_in_two_models INT;
  latest_model_value INT := 200000;
  old_model_value INT := 100000;
BEGIN
  difference_in_two_models := latest_model_value - old_model_value;

  RETURN difference_in_two_models;
END;
$$ LANGUAGE plpgsql;
SELECT get_car_price(200000, 100000);
```

ALIAS

1>Assigning a Table Alias:
2>Using Table Aliases in Join Operations:
3>Table Aliases with Subqueries:

1>Assigning a Table Alias:-

Syntax:-

```
SELECT column_name
FROM table_name AS alias_name;
```

Example:-

```
postgres=# select name as full_name from google;
 full_name
------------
 roman
 sting
 andrew
 Thomas
 jordan
 Sun Jin Wo
(6 rows)
```

2>Using Table Aliases in Join Operations:-

Syntax:-

```
SELECT t1.column_name, t2.column_name
FROM table1 AS t1
INNER JOIN table2 AS t2 ON t1.id = t2.table1_id;
```

Example:-

```
postgres=# select g.name ,m.name
postgres-# from google as g
postgres-# inner join microsoft as m on g.id=m.id;
    name    |  name
------------+---------
 jordan     | raheem
 Sun Jin Wo | Michael
 andrew     | akbar
 Thomas     | rasheed
 roman      | imad
 sting      | wajahat
(6 rows)
```

3) Table Aliases with Subqueries:

Syntax:-

```
SELECT t1.column_name
FROM (
  SELECT column_name
  FROM table_name
) AS t1;
```

Example:-

```
postgres=# select t1.name,t1.age,t1.specialisation
postgres-# from(select name,age,specialisation from google)
postgres-# as t1;
   name    | age |      specialisation
-----------+-----+---------------------------
 roman     |  24 | Data Scientist
 sting     |  22 | Machine Learning Engineer
 andrew    |  26 | Data Analyst
 Thomas    |  42 | AI
 jordan    |  26 | Network Engineer
 Sun Jin Wo |  27 | hacker
(6 rows)
```

Trigger

-- Create the trigger function:-

To create a trigger function in PostgreSQL, you can use the `CREATE FUNCTION` statement.
Here's an example of creating a trigger function called `after_insert_trigger_function`

```
CREATE OR REPLACE FUNCTION after_insert_trigger_function()
RETURNS TRIGGER AS $$
BEGIN
    -- Trigger function logic goes here
    -- You can perform actions or execute SQL statements
    -- based on your specific requirements
    -- Use the NEW and OLD keywords to access the values of the affected rows

    -- Example: Raise a notice with the inserted row's ID
    RAISE NOTICE 'New row inserted with ID: %', NEW.id;

    -- Return the NEW row to complete the trigger function
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Create the trigger:-

To create a trigger in PostgreSQL, you can use the `CREATE TRIGGER` statement. Here's an example of creating a trigger called `after_insert_trigger` that executes the `after_insert_trigger_function` after an `INSERT` operation on a table named "employees":

```
CREATE TRIGGER after_insert_trigger
AFTER INSERT ON employees
FOR EACH ROW
EXECUTE FUNCTION after_insert_trigger_function();
```

1>`after_insert_trigger` is the name given to the trigger. You can choose a descriptive name that reflects the purpose of the trigger.
2>`AFTER INSERT` specifies that the trigger should be executed after an `INSERT` operation on the associated table.
3>`ON employees` specifies the table on which the trigger is defined. Replace `employees` with the actual name of your table.
4>`FOR EACH ROW` indicates that the trigger function will be executed for each affected row.
5>`EXECUTE FUNCTION after_insert_trigger_function()` specifies the trigger function to be executed when the trigger is fired. Replace `after_insert_trigger_function` with the actual name of your trigger function.

Syntax:-
Create the trigger function:-

```
CREATE OR REPLACE FUNCTION after_insert_trigger_function()
RETURNS TRIGGER AS $$
BEGIN
   RAISE NOTICE 'New row inserted with ID: %', NEW.id;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Create the trigger:-

```
CREATE TRIGGER after_insert_trigger
AFTER INSERT ON employees
FOR EACH ROW
EXECUTE FUNCTION after_insert_trigger_function();
```

Example:-
Create Function():-

```
CREATE OR REPLACE FUNCTION after_insert_trigger_function()
RETURNS TRIGGER AS $$
BEGIN
RAISE NOTICE 'New row inserted with ID: %', NEW.id;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Create Trigger:-

```
CREATE TRIGGER after_insert_trigger
after insert on google
for each row
EXECUTE FUNCTION after_insert_trigger_function();
```

Index

Syntax:-

CREATE INDEX index_name
ON table_name (column1, column2, ...);

```
CREATE INDEX index_name ON table_name [USING method]
(
   column_name [ASC | DESC] [NULLS {FIRST | LAST}],
   ...
);
```

[USING method] :-

- It is used to specify the index methods, such as **B-tree, GIN, HASH, GiST, BRIN, and SP-GiST.**
- By default, PostgreSQL uses **B-tree Index**.

Example:-

create index employees_information
on google (id,name,age,specialisation,salary);

```
postgres=# \d employees_information

           Index "public.employees_information"
    Column     |          Type          | Key? |   Definition
----------------+------------------------+------+----------------
 id            | integer                | yes  | id
 name          | character varying(255) | yes  | name
 age           | integer                | yes  | age
 specialisation | character varying(240) | yes  | specialisation
 salary        | integer                | yes  | salary
btree, for table "public.google"
```

__Example 2:-__
__USING HASH:-__

CREATE INDEX imaad ON google USING HASH (salary);

```
postgres=# \d google
                     Table "public.google"
   Column     |          Type          | Collation | Nullable | Default
---------------+------------------------+-----------+----------+---------
 id            | integer                |           |          |
 name          | character varying(255) |           |          |
 age           | integer                |           |          |
 specialisation | character varying(240) |           |          |
 salary        | integer                |           |          |
Indexes:
   "employees_information" btree (id, name, age, specialisation, salary)
    "imaad" hash (salary)
```

__Example 3 :-__
__Using BRIN:-__

CREATE INDEX hydra on google using BRIN(salary);

```
postgres=# \d hydra
       Index "public.hydra"
 Column |  Type   | Key? | Definition
--------+---------+------+------------
 salary | integer | yes  | salary
brin, for table "public.google"
```