PostgreSQL Functions

What is a Function in PostgreSQL?

- A **function** (or stored procedure) is a reusable block of code stored on the database server.
- It can contain **SQL statements + procedural code** (loops, conditions, variables).
- Functions save time because instead of running multiple queries, you call the function once.
- PostgreSQL supports functions in many languages: **SQL, PL/pgSQL, C, Python**, etc.

Note :Create a new database this work

CREATE DATABASE mydb;

CREATE FUNCTION Command

Syntax

```
CREATE [OR REPLACE] FUNCTION function_name (arguments)
RETURNS return_datatype
LANGUAGE plpgsql
AS $$
DECLARE
  -- variable declarations
BEGIN
  -- function logic
  RETURN value;
END;
$$;
```

Explanation of parts:

- **function_name** → Name of function.
- **OR REPLACE** → Updates existing function.
- **arguments** → Input parameters (can be none or many).
- **RETURNS** → The data type returned by the function.
- **LANGUAGE** → Programming language (e.g., plpgsql).
- **DECLARE** → Section to declare variables.
- **BEGIN … END** → Main logic of the function.
- **RETURN** → Final result of function.

Example – Function on Car Table

Suppose we have a table **Car** with a column **Car_price**.

We create a function to count cars within a price range:

```
CREATE FUNCTION get_car_Price(Price_from int, Price_to int)
RETURNS int
LANGUAGE plplgsql
AS $$
DECLARE
  Car_count integer;
BEGIN
  SELECT COUNT(*) INTO Car_count
  FROM Car
  WHERE Car_price BETWEEN Price_from AND Price_to;
```

```
  RETURN Car_count;
END;
$$;
```

How it works?

1. Function name → get_car_Price
2. Inputs → Price_from and Price_to
3. Inside function → it counts cars whose price is between given values.
4. Returns → number of cars (integer)

Structure of get_car_Price Function

1. **Header Section**
    - Function name → get_car_Price()
    - Parameters → Price_from INT, Price_to INT
    - Return type → INT
    - Language → plpgsql
2. **Function Body**
    - Written inside $$ … $$
    - **DECLARE** → variable Car_count
    - **BEGIN…END** →
        - SELECT INTO → counts cars between given price range
        - RETURN → returns Car_count

PostgreSQL – Creating Functions

1. Ways to Create a Function

- **pgAdmin** (GUI tool)
  - Open pgAdmin → connect DB → Query Tool → write function code → Execute → Refresh functions list.

- **SQL Shell (psql)** (command line)
  - Connect to DB: \c database_name
  - Write function code in shell.
  - List all functions: \df

Create Function in PostgreSQL using pgAdmin

1) Open pgAdmin

- Launch **pgAdmin 4** from your system.
- Log in with your PostgreSQL user (default: postgres).

2) Connect to the Server

- Expand **Servers** → **PostgreSQL 16 (or your version)**.
- Enter the password if asked.

3) Select / Create Database

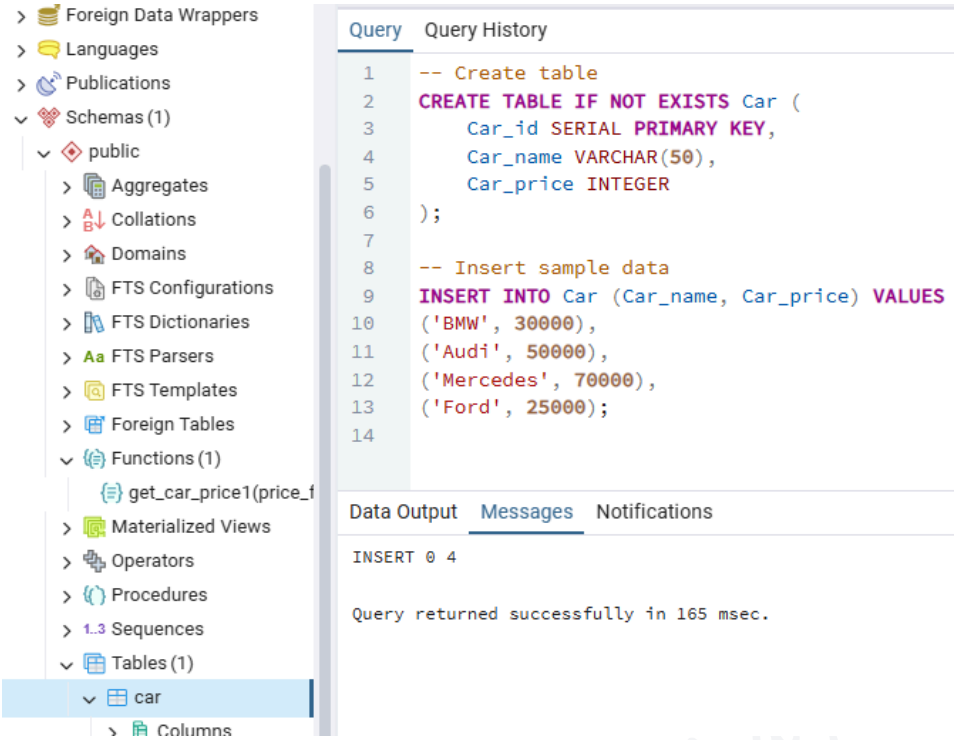
- Expand **Databases**.

- If you already created javatpoint (from earlier), right-click → **Connect**.
- If not:
    - Right-click **Databases** → **Create** → **Database...**
    - Enter name mydb→ **Save**.

4) Create Table Car (if not already created)

- Expand your database → **Schemas** → **public** → **Tables**.
- Right-click **Tables** → **Create** → **Table...**
-

```
-- Create table
CREATE TABLE IF NOT EXISTS Car (
    Car_id SERIAL PRIMARY KEY,
    Car_name VARCHAR(50),
    Car_price INTEGER
);
-- Insert sample data
INSERT INTO Car (Car_name, Car_price) VALUES
('BMW', 30000),
('Audi', 50000),
('Mercedes', 70000),
('Ford', 25000);
```



- Click **Save**.

5) Create the Function

- Expand **mydb→ Schemas → public → Functions**.
- Right-click **Functions → Create → Function...**

Now fill in:

General Tab

- Name: get_car_price1

Definition Tab

- Return type: integer
- Language: plpgsql

Code Tab

```
DECLARE
  Car_count INT;
BEGIN
  SELECT COUNT(*) INTO Car_count
  FROM Car
  WHERE Car_price BETWEEN Price_from AND Price_to;
  RETURN Car_count;
END;
```

Arguments Tab

Click **+** and add parameters:

1. Name: Price_from → Type: integer
2. Name: Price_to → Type: integer

Click **Save**

6) Verify Function

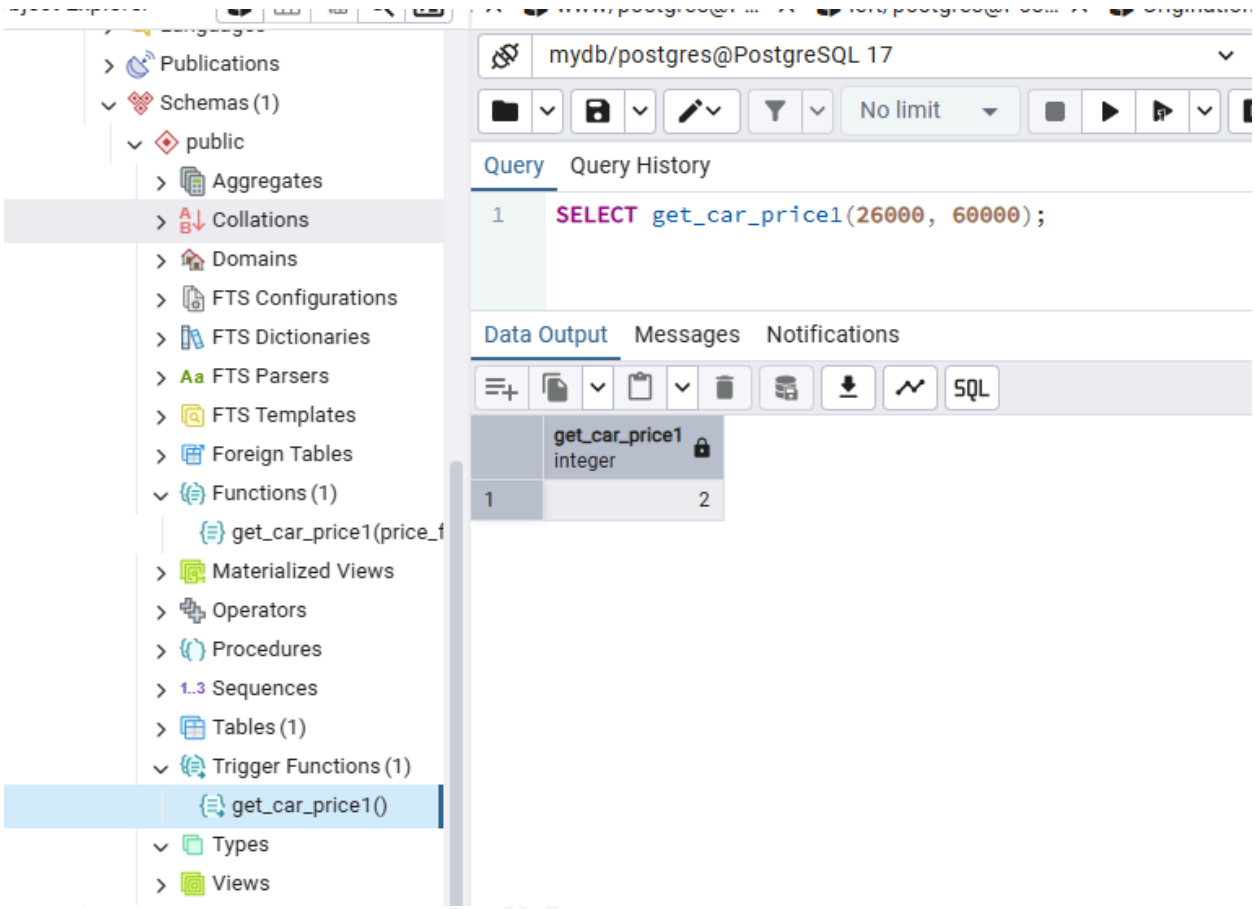
- Expand **Functions** → **get_car_price1** should now appear.
- Right-click → **Properties** to review.

7) Test the Function

- Open **Query Tool** (right-click DB → Query Tool).
- Run:

```
SELECT get_car_price1(26000, 60000);
```

Expected output:



8) Check Function List (Optional)

- In Query Tool:

\df

(or just see it under **Functions** tree in pgAdmin).

Create Function using SQL Shell (psql)

1) Open psql

Linux/macOS:

sudo -u postgres psql

or

psql -U postgres

Windows (SQL Shell):

we will open the psqlin our local system

(Enter the password if asked.)

2) (Optional) Create the database

If you need a fresh DB:

```
CREATE DATABASE mydb1;
```

Expected: CREATE DATABASE

3) Connect to the database

```
\c mydb1
```

4) Create the sample table Car

```
CREATE TABLE Car1 (
  Car_id SERIAL PRIMARY KEY,
  Car_name VARCHAR(50),
  Car_price INT
);
```

```
mydb1=# CREATE TABLE Car1 (
mydb1(#   Car_id SERIAL PRIMARY KEY,
mydb1(#   Car_name VARCHAR(50),
mydb1(#   Car_price INT
mydb1(# );
CREATE TABLE
mydb1=#
```

5) Insert sample data

```
INSERT INTO Car1 (Car_name, Car_price) VALUES
('BMW', 30000),
('Audi', 50000),
('Mercedes', 70000),
('Ford', 25000);
```

```
mydb1=# INSERT INTO Car1 (Car_name, Car_price) VALUES
mydb1-# ('BMW', 30000),
mydb1-# ('Audi', 50000),
mydb1-# ('Mercedes', 70000),
mydb1-# ('Ford', 25000);
INSERT 0 4
mydb1=#
```

6) Verify data (optional)

```
SELECT * FROM Car1;
```

```
mydb1=# SELECT * FROM Car1;
 car_id | car_name | car_price
--------+----------+-----------
      1 | BMW      |     30000
      2 | Audi     |     50000
      3 | Mercedes |     70000
      4 | Ford     |     25000
(4 rows)


mydb1=#
```

7) Create the function (corrected & safe)

```
CREATE OR REPLACE FUNCTION get_car_price1(Price_from INT, Price_to INT)

RETURNS INT

LANGUAGE plpgsql

AS $$

DECLARE

  Car_count INT;

BEGIN

  SELECT COUNT(*) INTO Car_count

  FROM Car1

  WHERE Car_price BETWEEN Price_from AND Price_to;


  RETURN Car_count;
```

END;

$$;

```
mydb1=# CREATE OR REPLACE FUNCTION get_car_price1(Price_from INT, Price_to INT)
mydb1-# RETURNS INT
mydb1-# LANGUAGE plpgsql
mydb1-# AS $$
mydb1$# DECLARE
mydb1$#   Car_count INT;
mydb1$# BEGIN
mydb1$#   SELECT COUNT(*) INTO Car_count
mydb1$#   FROM Car1
mydb1$#   WHERE Car_price BETWEEN Price_from AND Price_to;
mydb1$#
mydb1$#   RETURN Car_count;
mydb1$# END;
mydb1$# $$;
CREATE FUNCTION
mydb1=#
```

Notes:

- Use CREATE OR REPLACE to avoid "already exists" errors.
- Use $$ ... $$ for the body.
- Ensure you RETURN Car_count; (not a typo like Price_count).

8) List / inspect the function

\df get_car_price1

-- or for more details:

\df+ get_car_price1

Shows function name, argument types, return type, language, owner, source.

```
mydb1=# \df get_car_price1
                                  List of functions
 Schema |      Name      | Result data type |         Argument data types          | Type
--------+----------------+------------------+--------------------------------------+------
 public | get_car_price1 | integer          | price_from integer, price_to integer | func
(1 row)


mydb1=#
```

9) Call / test the function

SELECT get_car_price1(26000, 60000);

```
mydb1=# SELECT get_car_price1(26000, 60000);
 get_car_price1
----------------
              2
(1 row)


mydb1=#
```

(30000 and 50000 fall inside the range → 2)

Named:

SELECT get_car_price1(Price_from => 26000, Price_to => 60000);

Mixed (positional first):

SELECT get_car_price1(26000, Price_to => 60000);

10) Common fixes & helpful commands

If function exists and you want to replace it:

CREATE OR REPLACE FUNCTION ...

To remove:

```
DROP FUNCTION IF EXISTS get_car_price1(INT, INT);
```

If relation "Car" does not exist → check \dt and ensure you are in the correct database/schema.

If you get syntax errors → check AS $$ ... $$; and semicolons.

If permission denied → run as a superuser or grant appropriate rights.

11) Exit psql

```
\q
```


Download Complete SQL & PostgreSQL Notes + Practice Files

You can access the full set of **SQL/PostgreSQL notes** along with practice datasets and queries from this GitHub repository:

👉 **SQL-resources-and-tutorials by akshay-dhage**