# Functions

## Definition:

In PostgreSQL, functions are used to encapsulate a series of operations into reusable blocks of code. They can be either built-in functions (provided by PostgreSQL) or user-defined functions (created by the user to meet specific requirements).

## Types of Functions in PostgreSQL:

1. **Built-in Functions**
2. **User-defined Functions**

## 1. Built-in Functions:

PostgreSQL comes with a rich set of built-in functions that allow you to perform various operations like mathematical computations, string manipulation, date/time processing, and more. These functions are predefined by PostgreSQL and do not require the user to define them.

**Examples of Built-in Functions:**

**1. Mathematical Functions:**

- abs(x)

  Description: Returns the absolute value of x.

  Example:  SELECT abs(-15);          -- Result: 15

- round(x [, d])

  Description: Rounds x to d decimal places (defaults to 0).

  Example:  SELECT round(123.4567, 2);     -- Result: 123.46

- ceil(x) or ceiling(x)

  Description: Returns the smallest integer greater than or equal to x.

  Example:  SELECT ceil(123.45);     -- Result: 124

- floor(x)

  Description: Returns the largest integer less than or equal to x.

  Example: SELECT floor(123.45);     -- Result: 123

- sqrt(x)

  Description: Returns the square root of x.

  Example: SELECT sqrt(25);              -- Result: 5

- random()

  Description: Returns a random number between 0 and 1.

  Example: SELECT random();

- power(x, y)

  Description: Returns x raised to the power of y.

  Example: SELECT power(2, 3);      -- Result: 8

**2. String Functions:**

- length(string)

  Description: Returns the length of the string.

  Example: SELECT length('Hello, World!');              -- Result: 13

- lower(string)

  Description: Converts the string to lowercase.

Example:  SELECT lower('PostgreSQL');    -- Result: postgresql

- upper(string)

  Description: Converts the string to uppercase.

  Example:  SELECT upper('PostgreSQL');   -- Result: POSTGRESQL

- concat(string1, string2, ...)

  Description: Concatenates two or more strings.

  Example:  SELECT concat('Hello, ', 'World!');         -- Result: Hello, World!

- substring(string FROM start FOR length)

  Description: Extracts a substring from a string.

  Example:  SELECT substring('PostgreSQL' FROM 1 FOR 4);       -- Result: Post

- - trim([leading | trailing | both] string)

  Description: Removes the specified characters (default is spaces) from the start and/or end of a string.

  Example:  SELECT trim(both ' ' FROM '  PostgreSQL  ');    -- Result: PostgreSQL

- replace(string, from, to)

  Description: Replaces all occurrences of from with to in the string.

  Example:  SELECT replace('Hello, World!', 'World', 'PostgreSQL');  -- Result: Hello, PostgreSQL!

- position(substring IN string)

Description: Returns the position of the first occurrence of substring in string.

Example:  SELECT position('SQL' IN 'PostgreSQL');          -- Result: 9

**3. Date/Time Functions**

- current_date

  Description: Returns the current date.

  Example:  SELECT current_date;    -- Result: 2024-12-04 (current date)

- current_time

  Description: Returns the current time (without date).

  Example:  SELECT current_time;     -- Result: 14:30:00 (current time)

- current_timestamp or now()

  Description: Returns the current date and time (timestamp with time zone).

  Example:  SELECT current_timestamp;      -- Result: 2024-12-04 14:30:00

- date_part(field, source)

  Description: Extracts a part of a date or timestamp (e.g., year, month, day).

  Example:  SELECT date_part('year', current_timestamp);             -- Result: 2024

- date_trunc(field, source)

  Description: Truncates a timestamp to the specified unit (e.g., 'year', 'month').

  Example:  SELECT date_trunc('month', current_timestamp);         -- Result: 2024-12-01 00:00:00

- extract(field FROM source)

  Description: Extracts a field (such as year, month, day) from a date or timestamp.

  Example:  SELECT extract(year FROM current_timestamp);                    -- Result: 2024

- to_char(source, format)

  Description: Formats a date, time, or timestamp into a string according to the specified format.

  Example:  SELECT to_char(current_timestamp, 'YYYY-MM-DD HH24:MI:SS');   -- Result: 2024-12-04 14:30:00

**4. Aggregate Functions**

- avg(column)

  Description: Returns the average value of a numeric column.

  Example:  SELECT avg(salary) FROM employees;

- count(*)

  Description: Returns the number of rows in a table.

  Example:  SELECT count(*) FROM employees;

- max(column)

  Description: Returns the maximum value in a column.

  Example:  SELECT max(salary) FROM employees;

- min(column)

  Description: Returns the minimum value in a column.

Example:  SELECT min(salary) FROM employees;

- sum(column)

  Description: Returns the sum of values in a column.

  Example:  SELECT sum(salary) FROM employees;

**5. Conditional Functions**

- coalesce(value1, value2, ...)

  Description: Returns the first non-null value in the list.

  Example:  SELECT coalesce(NULL, 'Hello', 'World');                    -- Result: Hello

- nullif(value1, value2)

  Description: Returns NULL if the two values are equal, otherwise returns the first value.

  Example:  SELECT nullif(5, 5);                    -- Result: NULL

- case

  Description: A conditional expression (similar to an IF-ELSE statement).

  Example:

```
SELECT
  CASE
    WHEN salary > 100000 THEN 'High Salary'
    ELSE 'Low Salary'
  END
FROM employees;
```

- greatest(value1, value2, ...)

  Description: Returns the largest value from a list of expressions.

  Example: SELECT greatest(2, 5, 3, 8); -- Result: 8

- least(value1, value2, ...)

  Description: Returns the smallest value from a list of expressions.

  Example: SELECT least(2, 5, 3, 8); -- Result: 2

**6. Type Conversion Functions**

- cast(expression AS type) or ::type

  Description: Converts an expression from one data type to another.

  Example: SELECT '2024-12-04'::date; -- Result: 2024-12-04

- to_number(string, format)

  Description: Converts a string to a number based on a given format.

  Example: SELECT to_number('123.45', '999.99'); -- Result: 123.45

- to_char(value, format)

  Description: Converts a value to a string using a specified format.

  Example: SELECT to_char(12345.6789, '99999.99'); -- Result: 12345.68

## 2. <u>User-defined Functions (UDFs):</u>

User-defined functions (UDFs) allow you to extend PostgreSQL's functionality by creating custom functions to meet your specific needs. These functions are written by the user and can be written in several languages supported by PostgreSQL, such as PL/pgSQL, PL/Python, PL/Perl, SQL, etc.

**Types of User-defined Functions:**

- **PL/pgSQL (PostgreSQL's procedural language)**: This is the most common language used for creating user-defined functions in PostgreSQL. It is a procedural language similar to Oracle's PL/SQL, designed for writing complex functions and stored procedures.
- **SQL Functions**: These are functions written directly in SQL. These functions are typically simple and use standard SQL operations.
- **PL/Python, PL/Perl, etc.**: PostgreSQL supports various programming languages for creating UDFs, enabling you to write functions in those languages if you need more advanced functionality.

**Examples of User-defined Functions:**

**(a) PL/pgSQL Function Example:**

This example defines a function calculate_bonus to compute an employee's bonus based on their salary.

```
-- Create a PL/pgSQL function to calculate bonus
CREATE OR REPLACE FUNCTION calculate_bonus(salary DECIMAL)
RETURNS DECIMAL AS $$
DECLARE
   bonus DECIMAL;
BEGIN
   IF salary < 50000 THEN
      bonus := salary * 0.05;  -- 5% bonus for salary < 50k
   ELSE
      bonus := salary * 0.10;  -- 10% bonus for salary >= 50k
   END IF;
   RETURN bonus;
END;
$$ LANGUAGE plpgsql;

-- Usage
SELECT calculate_bonus(40000);
-- Result: 2000

SELECT calculate_bonus(60000);
-- Result: 6000
```

**Explanation:**

- **Function Name (calculate_bonus)**: The function takes a salary as input and returns the calculated bonus based on the salary.
- **Logic**: If the salary is below $50,000, a 5% bonus is given; otherwise, a 10% bonus is awarded.
- **Return**: The function returns the bonus amount.

**(b) SQL Function Example:**

In this example, a simple SQL function computes the square of a number.

```
-- Create an SQL function to calculate the square of a number
CREATE OR REPLACE FUNCTION square(n INT)
RETURNS INT AS $$
SELECT n * n;
$$ LANGUAGE sql;

-- Usage
SELECT square(4);
-- Result: 16
```

**Explanation:**

- **Function Name (square)**: This function calculates the square of an integer n by returning n * n.
- **Language**: This function is written entirely in SQL and uses the SQL SELECT statement to perform the calculation.

**(c) PL/Python Function Example:**

Here’s a simple example of a function written in Python (using the PL/Python extension) to return the factorial of a number.

```
-- Create a PL/Python function to calculate the factorial of a number
CREATE OR REPLACE FUNCTION factorial(n INT)
RETURNS INT AS $$
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
$$ LANGUAGE plpython3u;

-- Usage
SELECT factorial(5);
```

-- Result: 120

**Explanation:**

- **Function Name (factorial)**: This function computes the factorial of a number n recursively using Python.
- **Language**: The function is written in Python (PL/Python). The plpython3u language handler is used for Python 3.

## Key Differences Between Built-in and User-defined Functions:

| Feature | Built-in Functions | User-defined Functions |
|---|---|---|
| **Definition** | Predefined by PostgreSQL | Defined by the user according to custom needs |
| **Usage** | Ready to use in queries (e.g., abs(), now()) | Created by the user using languages like PL/pgSQL, SQL, PL/Python, etc. |
| **Flexibility** | Limited to the functionality provided by PostgreSQL | Highly flexible; user can create any custom logic |
| **Performance** | Optimized by PostgreSQL | May require performance tuning depending on complexity |
| **Language** | No programming language required | Written in PostgreSQL's languages (PL/pgSQL, PL/Python, etc.) |

# Procedures

## Definition:

A procedure in PostgreSQL is also a set of SQL or procedural code that performs a specific task, but it does not return a value. Procedures are defined using the CREATE PROCEDURE statement. They can take input (and output) parameters, and they are typically used to perform actions like modifying data or handling transactions, rather than returning values.

## Purpose:

- **Side Effects**: Procedures are generally used for performing operations that affect the database (like inserts, updates, deletes) or handle transactional logic.
- **No Return Value**: Unlike functions, procedures don't return a value. Instead, they are designed to execute some operation.
- **Complex Operations**: Procedures are suitable for operations that involve multiple steps, like transactions or error handling, or actions that need to change data but don't require a return result.
- **Transaction Control**: Procedures can commit or roll back transactions, which is a major difference from functions.

## Key Characteristics of Procedures:

- No return value.
- Typically used for performing data modifications or executing business logic with side effects.
- Can execute multiple SQL statements (not just return a result).
- Can commit or rollback transactions, which functions cannot do.
- Cannot be used directly in queries (they are invoked using the CALL statement).

## Basic Syntax of Creating a Procedure:

```
CREATE PROCEDURE procedure_name (parameter1 data_type, parameter2 data_type, ...)
LANGUAGE plpgsql
AS $$
BEGIN
   -- procedure logic here (no return value)
END;
$$;
```

## Procedure Parameters:

- procedure_name: Name of the procedure.
- parameter1, parameter2, ...: Input parameters (optional).

- No RETURNS clause: Procedures do not return a value.
- plpgsql: Indicates that the procedure uses PL/pgSQL.

## Sample Procedure Examples:

**Example 1: Simple Procedure for Inserting Data**

This procedure inserts a new employee into the employees table.

```
CREATE OR REPLACE PROCEDURE insert_employee(emp_name TEXT, emp_salary
DECIMAL)
AS $$
BEGIN
   INSERT INTO employees (name, salary) VALUES (emp_name, emp_salary);
END;
$$ LANGUAGE plpgsql;

-- Usage
CALL insert_employee('John Doe', 55000);
```

Explanation:
- The procedure insert_employee takes two parameters: emp_name (name of the employee) and emp_salary (salary of the employee).
- It performs an INSERT operation to add the new employee to the employees table.
- The procedure doesn't return a value, unlike a function.

**Example 2: Procedure with Error Handling**

This procedure updates the salary of an employee. It checks if the employee exists before performing the update, and raises an error if the employee is not found.

```
CREATE OR REPLACE PROCEDURE update_employee_salary(emp_id INT, new_salary
DECIMAL)
AS $$
BEGIN
   UPDATE employees
   SET salary = new_salary
   WHERE id = emp_id;

   IF NOT FOUND THEN
      RAISE EXCEPTION 'Employee with ID % not found', emp_id;
   END IF;
END;
$$ LANGUAGE plpgsql;

-- Usage
CALL update_employee_salary(1, 60000);  -- Updates salary for employee with ID 1
```

-- If an invalid ID is passed (e.g., an employee with ID 999 doesn't exist), an error will be raised.
CALL update_employee_salary(999, 60000);  -- Will raise an exception

Explanation:
- The procedure update_employee_salary takes two parameters: emp_id (the ID of the employee) and new_salary (the new salary).
- It performs an UPDATE on the employees table.
- The IF NOT FOUND statement checks if the update affected any rows. If no employee with the given ID exists, it raises an exception using RAISE EXCEPTION.

## **<u>Functions vs. Procedures in PostgreSQL:</u>**

| **Feature** | **Function** | **Procedure** |
|---|---|---|
| **Return Value** | Returns a single value or set | Does not return a value |
| **Used in Queries** | Can be used in SELECT, INSERT, UPDATE, etc. | Cannot be used directly in queries |
| **Transaction Control** | Cannot commit or rollback within function | Can commit or rollback changes |
| **SQL Statements** | Can execute SQL queries | Can execute SQL queries, manage transactions |
| **Declared with** | CREATE FUNCTION | CREATE PROCEDURE |
| **Used for** | Generally used for calculations or querying data | Generally used for data modification (insert, update, delete) |