I will give you an example of a simple **function** and **procedure** in PostgreSQL, along with explanations for each.

**Example 1: Simple Function in PostgreSQL**

Let’s create a simple function that calculates the **yearly bonus** for an employee based on their salary and a bonus percentage.

**DDL for employees Table**

sql

Copy code

CREATE TABLE employees (

employee\_id SERIAL PRIMARY KEY,

name VARCHAR(100),

salary NUMERIC

);

-- Insert sample data

INSERT INTO employees (name, salary) VALUES

('Alice Johnson', 50000),

('Bob Smith', 60000),

('Charlie Brown', 70000);

**Create a Simple Function to Calculate Yearly Bonus**

sql

Copy code

CREATE OR REPLACE FUNCTION calculate\_bonus(emp\_salary NUMERIC, bonus\_percent NUMERIC)

RETURNS NUMERIC AS $$

BEGIN

-- Calculate bonus as salary \* bonus percentage

RETURN emp\_salary \* (bonus\_percent / 100);

END;

$$ LANGUAGE plpgsql;

**Explanation**

* The function calculate\_bonus takes two arguments: emp\_salary (the employee’s salary) and bonus\_percent (the percentage bonus).
* It returns the calculated bonus as a NUMERIC value by multiplying the salary by the bonus percentage.
* The RETURN statement inside the function body computes and returns the bonus.

**Usage of the Function**

You can use this function in a query to calculate bonuses for employees:

sql

Copy code

SELECT name, salary, calculate\_bonus(salary, 10) AS yearly\_bonus

FROM employees;

**Example 2: Simple Procedure in PostgreSQL**

Let’s create a procedure that **increases the salary** of an employee by a given percentage.

**Create a Simple Procedure to Update Salary**

sql

Copy code

CREATE OR REPLACE PROCEDURE increase\_salary(emp\_id INT, increase\_percent NUMERIC)

LANGUAGE plpgsql AS $$

BEGIN

-- Update the salary of the employee by increasing it by the given percentage

UPDATE employees

SET salary = salary + (salary \* (increase\_percent / 100))

WHERE employee\_id = emp\_id;

END;

$$;

**Explanation**

* The procedure increase\_salary takes two arguments: emp\_id (the employee’s ID) and increase\_percent (the percentage increase).
* The UPDATE statement inside the procedure increases the salary by the given percentage.
* Unlike functions, procedures do not return values. Instead, they perform actions (in this case, updating data).

**Usage of the Procedure**

To increase the salary of an employee by 10%, you can call the procedure like this:

sql

Copy code

CALL increase\_salary(1, 10); -- Increases Alice Johnson's salary by 10%

You can then check the employees table to see the updated salary:

sql

Copy code

SELECT \* FROM employees;

**Final Example Output (Before and After Salary Update)**

| **employee\_id** | **name** | **salary** |
| --- | --- | --- |
| 1 | Alice Johnson | 50000 |
| 2 | Bob Smith | 60000 |
| 3 | Charlie Brown | 70000 |

After running CALL increase\_salary(1, 10);, Alice’s salary increases by 10%, so her new salary will be 55000.

| **employee\_id** | **name** | **salary** |
| --- | --- | --- |
| 1 | Alice Johnson | 55000 |
| 2 | Bob Smith | 60000 |
| 3 | Charlie Brown | 70000 |

**Summary**

* **Function**: Used to compute a value and return it (e.g., calculating a bonus).
* **Procedure**: Used to perform an action, like updating the salary, without returning any value.

Both of these constructs are extremely useful when you need to encapsulate logic and reuse it in your PostgreSQL database operations.

**Available RAISE Levels**

* RAISE NOTICE: Outputs informational messages. This is the most commonly used level for debugging or logging non-critical information.
* RAISE INFO: Similar to NOTICE, but with a lower priority (less visible by default).
* RAISE WARNING: Outputs warning messages that might indicate something unexpected, but doesn’t stop execution.
* RAISE EXCEPTION: Outputs error messages and stops the execution of the current transaction.