UNIVERSITY INSTITUTE *of*
COMPUTING
Asia's Fastest Growing University

NAAC GRADE A+

Student Name: Aayush Raj **UID: 25MCI10066**
Branch: M.C.A(Al & ML) **Section/Group: 25MAM-1 A**
Semester: 2nd **Date of Performance:03/02/2026**
Subject Name: TECHINCAL SKILLS **Subject Code: 25CAP - 652**

WORKSHEET 4

AIM: To understand and implement iterative control structures in PostgreSQL conceptually, including FOR loops, WHILE loops, and basic LOOP constructs, for repeated execution of database logic.

Tools Used: PostgreSQL

OBJECTIVES:

- To understand why iteration is required in database programming
- To learn the purpose and behavior of FOR, WHILE, and LOOP constructs
- To understand how repeated data processing is handled in databases
- To relate loop concepts to real-world batch processing scenarios
- To strengthen conceptual knowledge of procedural SQL used in enterprise systems

Given:

Practical / Experiment Steps

Step 1: FOR Loop – Simple Iteration

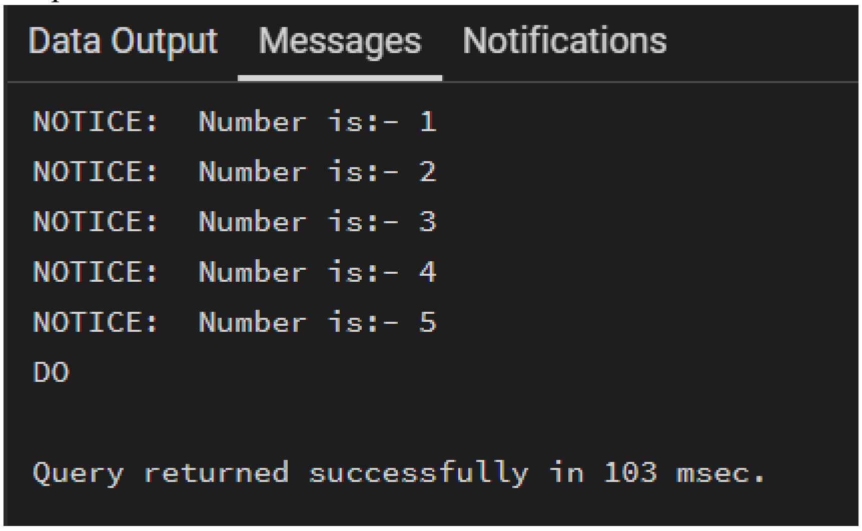
- **The loop runs a fixed number of times**
- **Each iteration represents one execution cycle**
- **Useful for understanding basic loop behavior**

Query:

```
DO $$
    DECLARE
        i INTEGER;
    BEGIN
        FOR i IN 1..5 LOOP
        RAISE NOTICE 'Number is:- %',i;

END LOOP;
END $$;
```

Output:



```
NOTICE:  Number is:- 1
NOTICE:  Number is:- 2
NOTICE:  Number is:- 3
NOTICE:  Number is:- 4
NOTICE:  Number is:- 5
DO

Query returned successfully in 103 msec.
```

Step 2: FOR Loop with Query (Row-by-Row Processing)

- **The loop processes database records one at a time**
- **Each iteration handles a single row**
- **Simulates cursor-based processing**

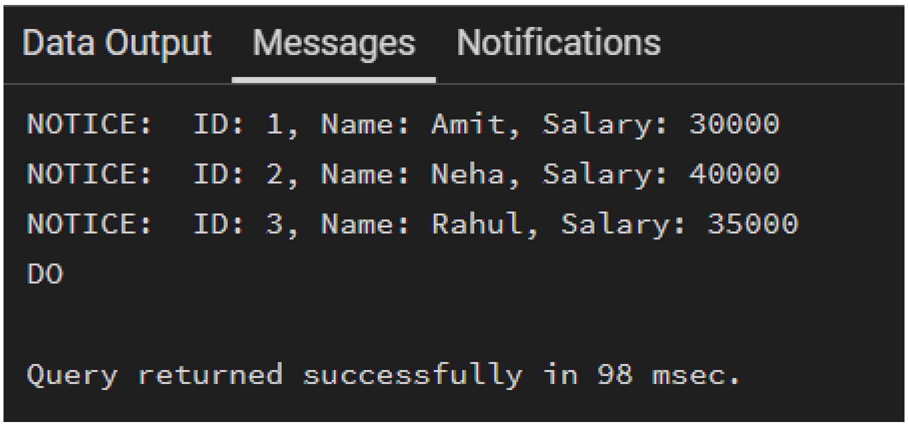
Query:

```
CREATE TABLE employee (
   emp_id INT,
   emp_name VARCHAR(50),
   salary INT
);


INSERT INTO employee VALUES
(1, 'Amit', 30000),
(2, 'Neha', 40000),
(3, 'Rahul', 35000);

DO $$
DECLARE
   rec RECORD;
BEGIN
   FOR rec IN SELECT * FROM employee LOOP
      RAISE NOTICE 'ID: %, Name: %, Salary: %',
      rec.emp_id, rec.emp_name, rec.salary;
   END LOOP;
END $$;
```

Output:



Data Output | Messages | Notifications

```
NOTICE:  ID: 1, Name: Amit, Salary: 30000
NOTICE:  ID: 2, Name: Neha, Salary: 40000
NOTICE:  ID: 3, Name: Rahul, Salary: 35000
DO

Query returned successfully in 98 msec.
```

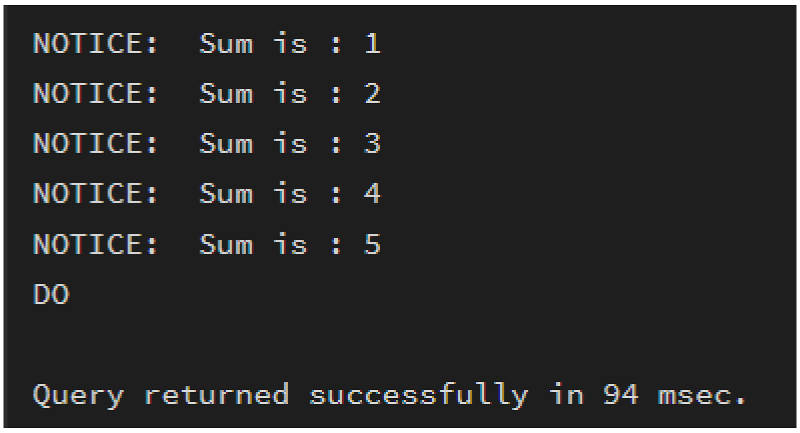
Step 3: WHILE Loop – Conditional Iteration

- **The loop runs until a condition becomes false**
- **Execution depends entirely on the condition**
- **The condition is checked before every iteration**

```
DO $$
DECLARE
        sum INTEGER := 1;
BEGIN
        WHILE sum <= 5 LOOP
                RAISE NOTICE 'Sum is : %',sum;
                sum := sum + 1;
        END LOOP;

END $$;
```

Output:



```
NOTICE:  Sum is : 1
NOTICE:  Sum is : 2
NOTICE:  Sum is : 3
NOTICE:  Sum is : 4
NOTICE:  Sum is : 5
DO

Query returned successfully in 94 msec.
```

Step 4: LOOP with EXIT WHEN

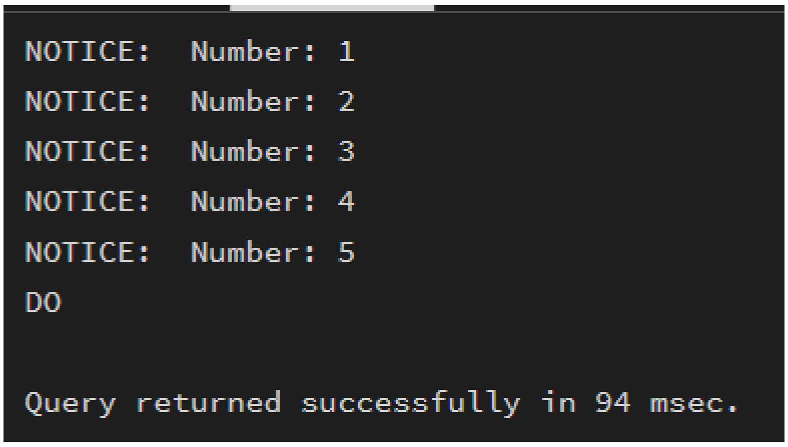
- **The loop does not stop automatically**
- **An explicit exit condition controls termination**
- **Gives flexibility in complex logic**

Query:

```
DO $$
DECLARE
   num INTEGER := 1;
BEGIN
   LOOP
      RAISE NOTICE 'Number: %', num;
      num := num + 1;

      EXIT WHEN num > 5;
   END LOOP;
END $$;
```

Output:-



```
NOTICE:  Number: 1
NOTICE:  Number: 2
NOTICE:  Number: 3
NOTICE:  Number: 4
NOTICE:  Number: 5
DO

Query returned successfully in 94 msec.
```

Step 5: Salary Increment Using FOR Loop

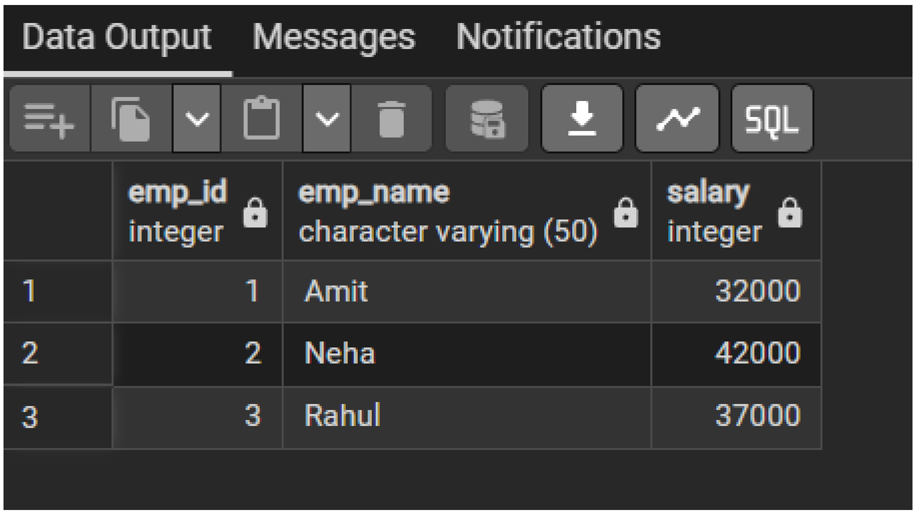
- **Employee records are processed one by one**
- **Salary values are updated iteratively**
- **Represents real-world payroll processing**

Query:

```
DO $$
DECLARE
   rec RECORD;
BEGIN
   FOR rec IN SELECT * FROM employee LOOP
      UPDATE employee
      SET salary = salary + 2000
      WHERE emp_id = rec.emp_id;
   END LOOP;
END $$;
```

```
SELECT * FROM employee;
```

Output:



Data Output | Messages | Notifications

	emp_id integer	emp_name character varying (50)	salary integer
1	1	Amit	32000
2	2	Neha	42000
3	3	Rahul	37000

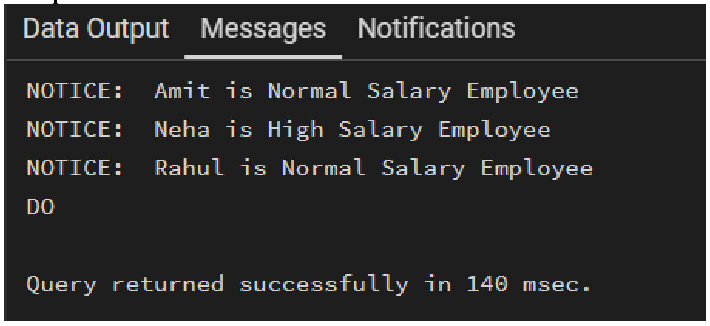
Step 6: Combining LOOP with IF Condition

- **Loop processes each record**
- **Conditional logic classifies data during iteration**
- **Demonstrates decision-making inside loops**

Query:

```
DO $$
DECLARE
  rec RECORD;
BEGIN
  FOR rec IN SELECT * FROM employee LOOP
    IF rec.salary >= 40000 THEN
      RAISE NOTICE '% is High Salary Employee', rec.emp_name;
    ELSE
      RAISE NOTICE '% is Normal Salary Employee', rec.emp_name;
    END IF;
  END LOOP;
END $$;
```

Output:



Data Output | Messages | Notifications

```
NOTICE:  Amit is Normal Salary Employee
NOTICE:  Neha is High Salary Employee
NOTICE:  Rahul is Normal Salary Employee
DO

Query returned successfully in 140 msec.
```

Learning Outcomes:

- Understand the **need for iterative control structures** in database programming.
- Explain the **syntax and working** of FOR, WHILE, and LOOP constructs in PostgreSQL.
- Implement **range-based and query-based FOR loops** using PL/pgSQL.
- Use **WHILE loops** for condition-controlled execution of SQL statements.
- Apply **LOOP with EXIT conditions** for flexible and custom termination logic.
- Perform **row-by-row data processing** inside PostgreSQL using procedural SQL.
- Integrate **conditional statements (IF–ELSE)** within loops for decision making.
- Understand how iterative logic is used in **real-world database applications** such as payroll processing, reporting, and batch operations.
- Gain foundational knowledge of **PL/pgSQL** used in enterprise-level database systems.