UNIVERSITY INSTITUTE of COMPUTING
Asia's Fastest Growing University

NAAC GRADE A+ ACCREDITED UNIVERSITY

Branch: MCA (Data Science)	**Semester:** 2nd
Student Name: Drishti Bhatia	**UID:** 25MCD10008
Subject Name: Technical Training - I Lab	**Subject Code**: 25CAP-652
Section/Group: 25MCD- KAR1	**Date of Performance:** 03-02-2026

Experiment No.: 4

Implementation of Iterative Control Structures using FOR, WHILE, and LOOP in PostgreSQL

1. **Aim of the practical:**
   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.

2. **Tools Used:** PostgreSQL

3. **Objectives of the practical:**
   - 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

4. **Theory:**
   In real-world database applications, tasks often need to be repeated multiple times. Examples include processing employee records, generating reports, validating data, applying salary increments, and running batch jobs. Standard SQL is declarative and works well for single operations, but repeated logic requires procedural control.
   PostgreSQL provides PL/pgSQL, a procedural extension that supports iteration using loop structures. These loops allow SQL statements to execute repeatedly until a specific condition is met.
   Iteration in PostgreSQL is commonly used inside:
   - Stored procedures
   - Functions
   - Anonymous execution blocks

   Large organizations such as Amazon, SAP, Oracle, and Rippling use loop-based logic for payroll processing, billing cycles, analytics, and automation workflows.

   Types of Loops in PostgreSQL

   1. FOR Loop (Range-Based)
   - Executes a fixed number of times
   - Useful when the number of iterations is known in advance
   - Commonly used for counters, testing, and batch execution

2. FOR Loop (Query-Based)
- Iterates over rows returned by a query
- Processes one row at a time
- Frequently used for reporting, audits, and row-wise calculations

3. WHILE Loop
- Executes repeatedly as long as a condition remains true
- Suitable for condition-controlled execution
- Often used in retry logic or threshold-based processing

4. LOOP with EXIT Condition
- Executes indefinitely until explicitly stopped
- Provides maximum control over execution flow
- Used in complex workflows where exit conditions are custom-defined

5. Experiment / Practical Steps:

Prerequisite Understanding

Example 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

Application: Counters, repeated tasks, batch execution

Example 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

Application: Employee reports, audits, data verification

Example 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

Application: Retry mechanisms, validation loops

Example 4: LOOP with EXIT WHEN
- The loop does not stop automatically
- An explicit exit condition controls termination
- Gives flexibility in complex logic

Application: Workflow engines, complex decision cycles

Example 5: Salary Increment Using FOR Loop
- Employee records are processed one by one
- Salary values are updated iteratively
- Represents real-world payroll processing

Application: Payroll systems, bulk updates

Example 6: Combining LOOP with IF Condition

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

Application: Employee grading, alerts, categorization logic

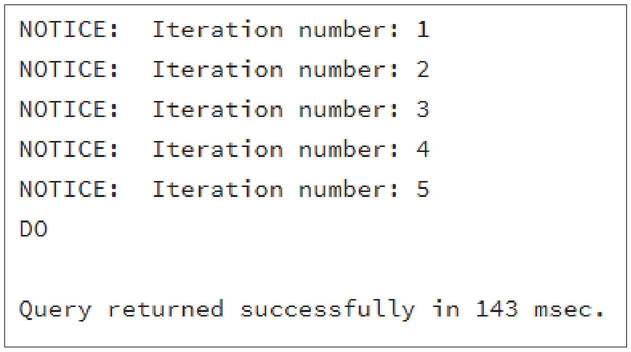
6. Code:

```
--Example1
DO $$
DECLARE
   i INT;
BEGIN
   FOR i IN 1..5 LOOP
      RAISE NOTICE 'Iteration number: %', i;
   END LOOP;
END $$;

--Example2
CREATE TABLE employee1 (
   emp_id SERIAL PRIMARY KEY,
   emp_name VARCHAR(50),
   department VARCHAR(30),
   salary INT
);

INSERT INTO employee1 (emp_name, department, salary) VALUES
('Aarav', 'HR', 45000),
('Diya', 'Finance', 60000),
('Kabir', 'IT', 75000),
('Meera', 'IT', 52000),
('Rohan', 'Marketing', 40000);

DO $$
DECLARE
   rec RECORD;
BEGIN
   FOR rec IN SELECT emp_id, emp_name FROM employee1 LOOP
      RAISE NOTICE 'Employee ID: %, Name: %', rec.emp_id, rec.emp_name;
   END LOOP;
END $$;

--Example3
DO $$
DECLARE
   counter INT := 1;
BEGIN
   WHILE counter <= 5 LOOP
```

```
        RAISE NOTICE 'Counter value: %', counter;
        counter := counter + 1;
    END LOOP;
END $$;

--Example4
DO $$
DECLARE
    num INT := 1;
BEGIN
    LOOP
        RAISE NOTICE 'Number: %', num;
        num := num + 1;

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

--Example5
DO $$
DECLARE
    rec RECORD;
BEGIN
    FOR rec IN SELECT emp_id FROM employee1 LOOP
        UPDATE employee1
        SET salary = salary + 1000
        WHERE emp_id = rec.emp_id;
    END LOOP;
END $$;

select * from employee1;

--Example6
DO $$
DECLARE
    rec RECORD;
BEGIN
    FOR rec IN SELECT emp_name, salary FROM employee1 LOOP
        IF rec.salary >= 60000 THEN
            RAISE NOTICE '% is a High Salary Employee', rec.emp_name;
        ELSE
            RAISE NOTICE '% is a Regular Salary Employee', rec.emp_name;
        END IF;
    END LOOP;
END $$;
```

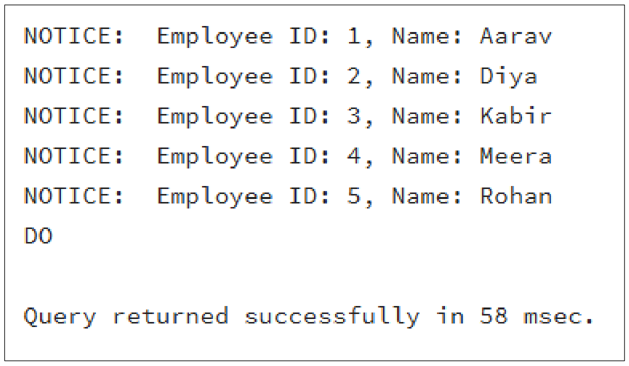
7. Output:

Example 1:



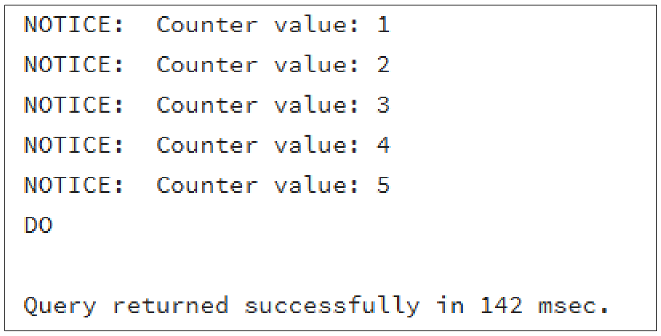
```
NOTICE:  Iteration number: 1
NOTICE:  Iteration number: 2
NOTICE:  Iteration number: 3
NOTICE:  Iteration number: 4
NOTICE:  Iteration number: 5
DO

Query returned successfully in 143 msec.
```

Example 2:



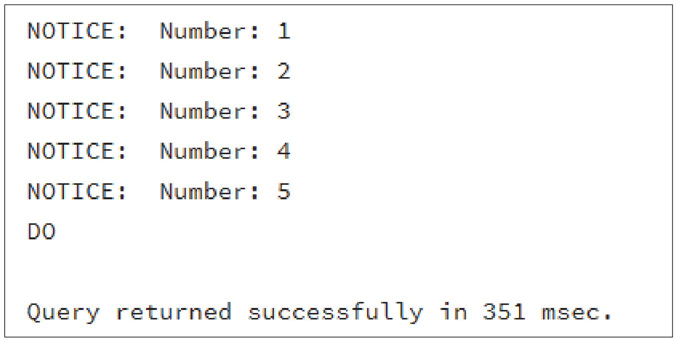
```
NOTICE:  Employee ID: 1, Name: Aarav
NOTICE:  Employee ID: 2, Name: Diya
NOTICE:  Employee ID: 3, Name: Kabir
NOTICE:  Employee ID: 4, Name: Meera
NOTICE:  Employee ID: 5, Name: Rohan
DO

Query returned successfully in 58 msec.
```

Example 3:



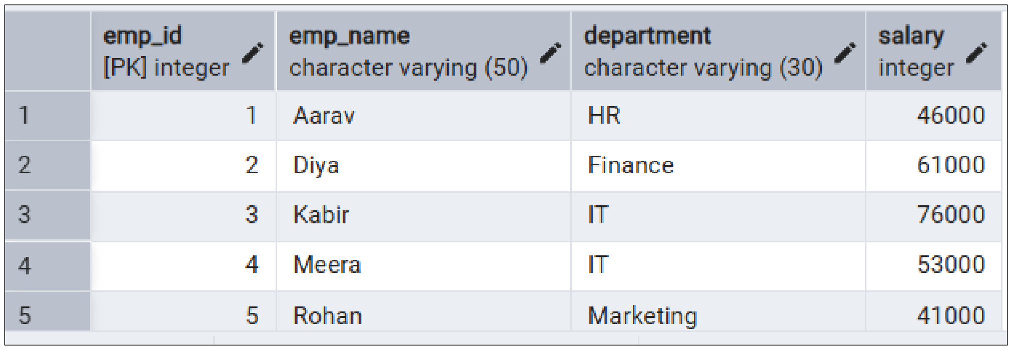
```
NOTICE:  Counter value: 1
NOTICE:  Counter value: 2
NOTICE:  Counter value: 3
NOTICE:  Counter value: 4
NOTICE:  Counter value: 5
DO

Query returned successfully in 142 msec.
```

Example 4:



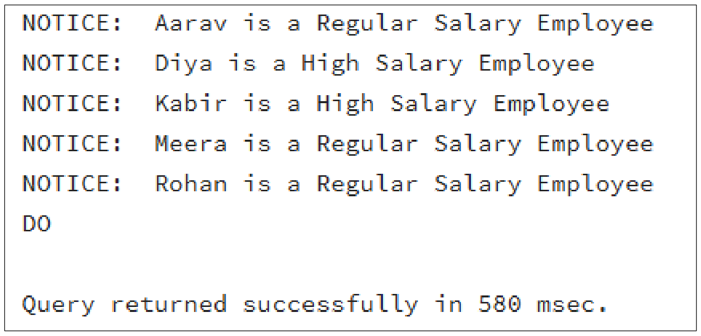
```
NOTICE:  Number: 1
NOTICE:  Number: 2
NOTICE:  Number: 3
NOTICE:  Number: 4
NOTICE:  Number: 5
DO

Query returned successfully in 351 msec.
```

Example 5:



	emp_id [PK] integer	emp_name character varying (50)	department character varying (30)	salary integer
1	1	Aarav	HR	46000
2	2	Diya	Finance	61000
3	3	Kabir	IT	76000
4	4	Meera	IT	53000
5	5	Rohan	Marketing	41000

Example 6:



```
NOTICE:  Aarav is a Regular Salary Employee
NOTICE:  Diya is a High Salary Employee
NOTICE:  Kabir is a High Salary Employee
NOTICE:  Meera is a Regular Salary Employee
NOTICE:  Rohan is a Regular Salary Employee
DO

Query returned successfully in 580 msec.
```

8. Learning Outcomes:

- Understood the concept of iterative control structures in PostgreSQL
- Identified appropriate use cases for FOR, WHILE, and LOOP constructs
- Implemented procedural logic using PL/pgSQL loops
- Applied loops for row-by-row data processing and conditional execution
- Gained foundational knowledge required for enterprise-level database applications