Worksheet No- 4

Name - Sahil Gupta	**UID – 25MCI10266**
Branch – MCA	**Section – MAM-1(A)**
Semester – 2nd	**Date of performance – 3rd feb,2026**
Subject - Technical Training	**Subject Code – 25CAP-652_25MAM_KAR-1_A**

Aim / Overview 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.

Software Requirements

- 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

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

Experiment Steps

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

RESULT:

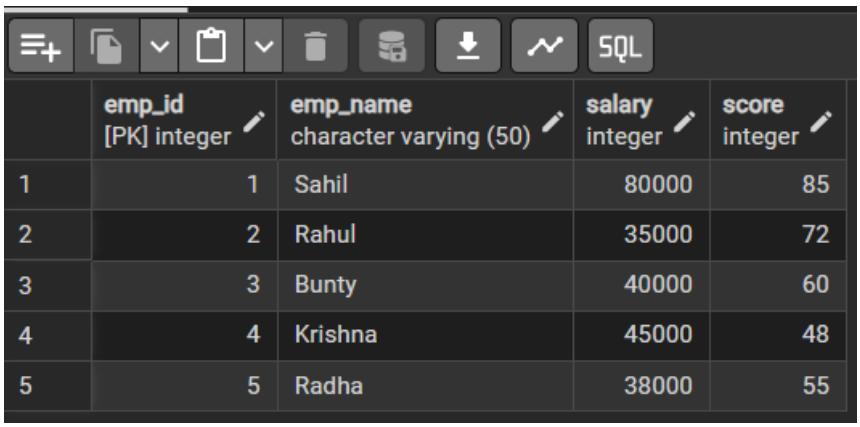
This experiment helps students understand how iterative control structures work in PostgreSQL at a conceptual level. Students learn where and why loops are used in database systems and gain foundational knowledge required for writing procedural logic in enterprise-grade applications.

Coding and Output :

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

INSERT INTO employee VALUES
(1, 'Sahil',   80000, 85),
(2, 'Rahul',     35000, 72),
(3, 'Bunty', 40000, 60),
(4, 'Krishna',   45000, 48),
(5, 'Radha',   38000, 55
```

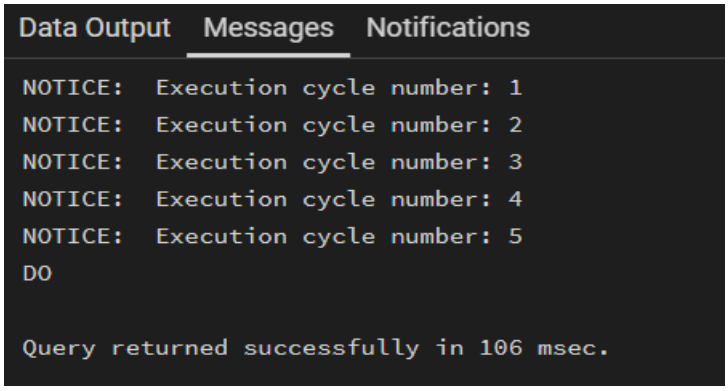
select * from employee;



	emp_id [PK] integer	emp_name character varying (50)	salary integer	score integer
1	1	Sahil	80000	85
2	2	Rahul	35000	72
3	3	Bunty	40000	60
4	4	Krishna	45000	48
5	5	Radha	38000	55

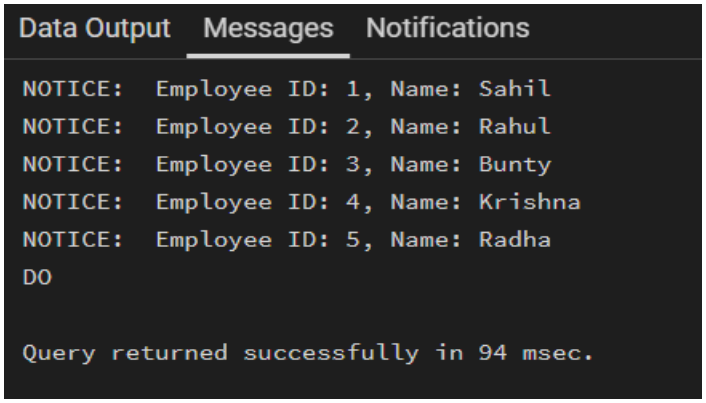
-- EXAMPLE 1 : For loop (Simple Iteration)

```
Do $$
Declare
        i INT;
Begin
        For i in 1..5 LOOP
                Raise notice 'Execution cycle number: %',i;
        end LOOP;
END $$;
```



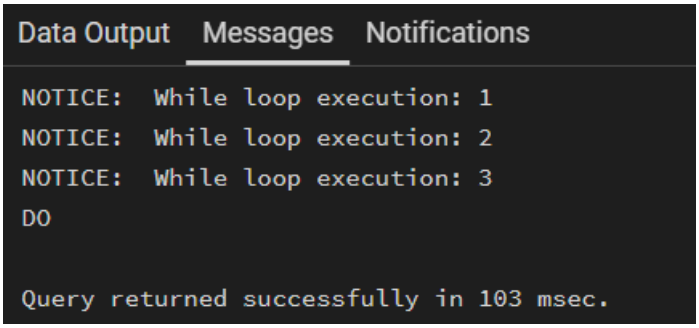
-- Example 2: For loop with query(Row-by-Row processing)

```
Do $$
Declare
        rec RECORD;
Begin
        For rec in select emp_id, emp_name from employee LOOP
                Raise notice 'Employee ID: %, Name: %',rec.emp_id,rec.emp_name;
        End LOOP;
End $$;
```



-- Example 3 : While LOOP - Conditional Iteration

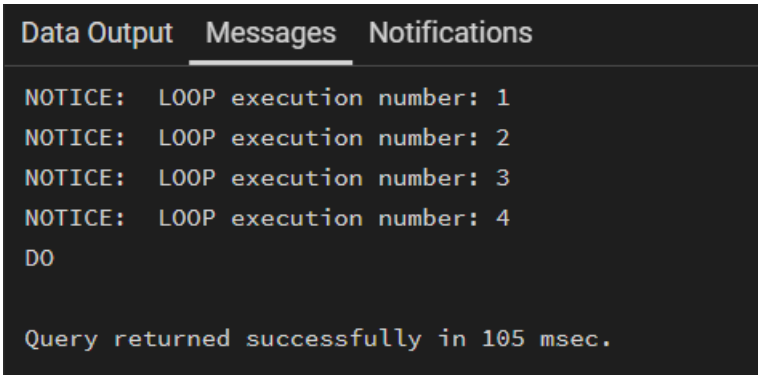
```
Do $$
Declare
        counter INT := 1;
Begin
        WHILE counter <= 3 LOOP
                Raise Notice 'While loop execution: %',counter;
                counter := counter + 1 ;
        End LOOP;
End $$;
```



-- Example 4 : LOOP with EXIT WHEN

```
Do $$
Declare
        i INT := 1;
Begin
        LOOP
                Raise notice 'LOOP execution number: %',i;
                i := i+1;

                Exit WHEN i > 4;
        END LOOP;
END $$;
```



-- Example 5: Salary increment Using For LOOP

```
Do $$
Declare
        rec RECORD;
Begin
        For rec in select emp_id FROM employee LOOP
                UPDATE employee
                SET salary = salary + 5000
                WHERE emp_id = rec.emp_id;
```

```
            Raise notice 'Salary incremented for Employee ID %', rec.emp_id;
        END LOOP;
END $$;
```

Data Output | Messages | Notifications

	emp_id [PK] integer	emp_name character varying (50)	salary integer	score integer
1	1	Sahil	85000	85
2	2	Rahul	40000	72
3	3	Bunty	45000	60
4	4	Krishna	50000	48
5	5	Radha	43000	55

This table shows the updated value of salary by 5000

-- Example 6: Combining LOOP and IF Condition

```
Do $$
Declare
        rec RECORD;
Begin
        For rec in select emp_name, score From employee LOOP

                IF rec.score >= 75 THEN
                        Raise NOTICE '% -> Grade: Distinction', rec.emp_name;
                ELSIF rec.score >= 60 THEN
                        Raise NOTICE '% -> Grade: First Class', rec.emp_name;
                ELSE
                        Raise NOTICE '% -> Grade: Pass', rec.emp_name;
                End if;
        End LOOP;
End $$;
```

Data Output | Messages | Notifications

```
NOTICE:  Sahil -> Grade: Distinction
NOTICE:  Rahul -> Grade: First Class
NOTICE:  Bunty -> Grade: First Class
NOTICE:  Krishna -> Grade: Pass
NOTICE:  Radha -> Grade: Pass
DO

Query returned successfully in 110 msec.
```