WORKSHEET 4

Student Name: Aradhya Sharma
UID: 25MCC20042
Branch: MCA (Cloud Computing & Devops)
Section/Group: 25MCC-101A
Semester: 2
Date of Performance: 03/02/2026
Subject Name: Technical Training 1
Subject Code: 25CAP-652

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.

S/W Requirement:

To perform this experiment, the following software is required:

1. **Operating System**
   - Windows 10 / 11, Linux, or macOS
2. **Database Management System (DBMS)**
   - PostgreSQL (version 12 or higher)
3. **SQL Client / Interface**
   - pgAdmin 4 (for PostgreSQL)

     OR
   - Command Line Interface (psql)

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

Practical / Experiment Steps

• Iterative control structures in PL/pgSQL to understand repeated execution of logic.

• A simple FOR loop was executed to observe fixed-count iteration behavior.

• A FOR loop with a SELECT query was used to process records row by row.

• A WHILE loop was implemented to demonstrate condition-based iteration.

• A basic LOOP with EXIT WHEN was executed to control manual termination.

• Employee salary records were updated iteratively using a FOR loop.

• Conditional logic was applied inside loops using IF–ELSE statements.

• Outputs were verified using RAISE NOTICE statements.

Procedure for experiment

Example 1: FOR Loop – Simple Iteration

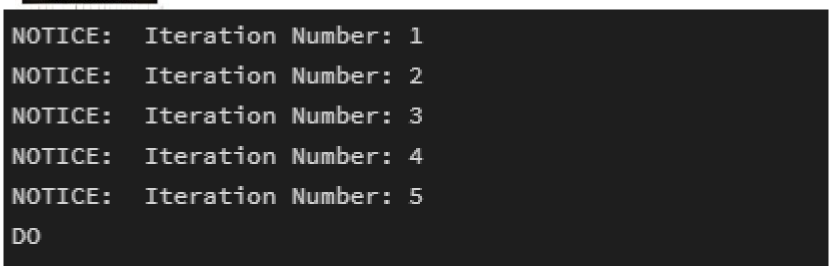
- The loop runs a fixed number of times
- Each iteration represents one execution cycle
- Useful for understanding basic loop behavior

Application: Counters, repeated tasks, batch execution

Query:

```
DO $$
BEGIN
   FOR i IN 1..5 LOOP
      RAISE NOTICE 'Iteration Number: %', i;
   END LOOP;
END $$;
```

Output:



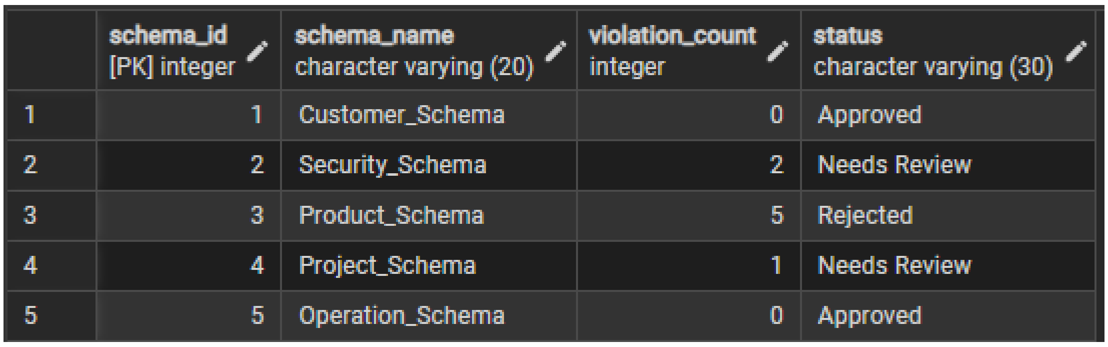
```
NOTICE:  Iteration Number: 1
NOTICE:  Iteration Number: 2
NOTICE:  Iteration Number: 3
NOTICE:  Iteration Number: 4
NOTICE:  Iteration Number: 5
DO
```

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

For table – violation_schema

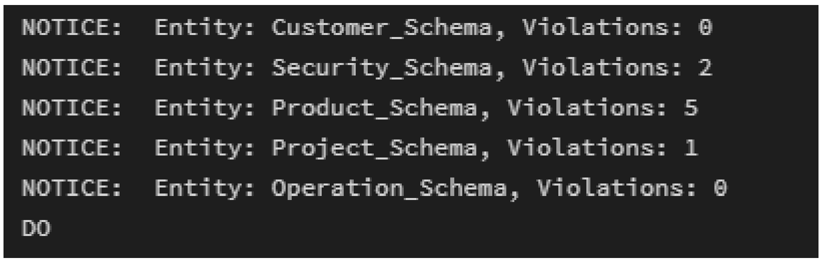


	schema_id [PK] integer	schema_name character varying (20)	violation_count integer	status character varying (30)
1	1	Customer_Schema	0	Approved
2	2	Security_Schema	2	Needs Review
3	3	Product_Schema	5	Rejected
4	4	Project_Schema	1	Needs Review
5	5	Operation_Schema	0	Approved

Query:

```
DO $$
DECLARE
   rec RECORD;
BEGIN
   FOR rec IN SELECT schema_name, violation_count FROM violation_schema LOOP
      RAISE NOTICE 'Entity: %, Violations: %',
                 rec.schema_name, rec.violation_count;
   END LOOP;
END $$;
```

Output:



```
NOTICE:  Entity: Customer_Schema, Violations: 0
NOTICE:  Entity: Security_Schema, Violations: 2
NOTICE:  Entity: Product_Schema, Violations: 5
NOTICE:  Entity: Project_Schema, Violations: 1
NOTICE:  Entity: Operation_Schema, Violations: 0
DO
```

Example 3: WHILE Loop – Conditional Iteration

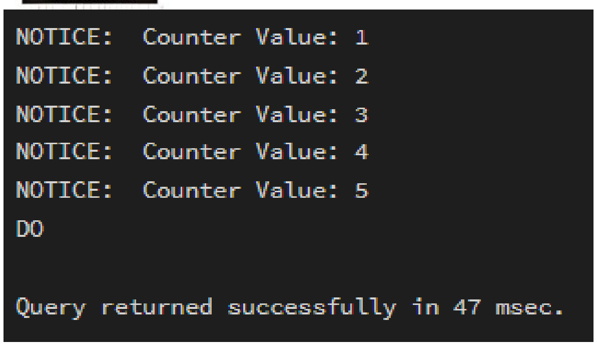
- The loop runs until a condition becomes false
- Execution depends entirely on the condition
- The condition is checked before every iteration

Application: Retry mechanisms, validation loops

Query:

```
DO $$
DECLARE
   counter INT := 1;
BEGIN
   WHILE counter <= 5 LOOP
      RAISE NOTICE 'Counter Value: %', counter;
      counter := counter + 1;
   END LOOP;
END $$;
```

Output:



```
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 47 msec.
```

Example 4: LOOP with EXIT WHEN

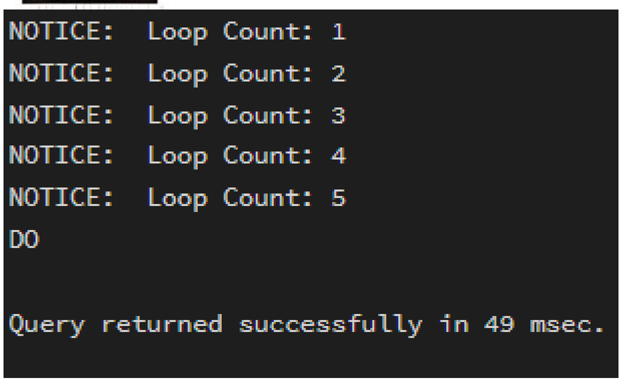
- The loop does not stop automatically
- An explicit exit condition controls termination
- Gives flexibility in complex logic

Application: Workflow engines, complex decision cycles

Query:

```
DO $$
DECLARE
   counter INT := 1;
BEGIN
   LOOP
      RAISE NOTICE 'Loop Count: %', counter;
      counter := counter + 1;
      EXIT WHEN counter > 5;
   END LOOP;
END $$;
```

Output:



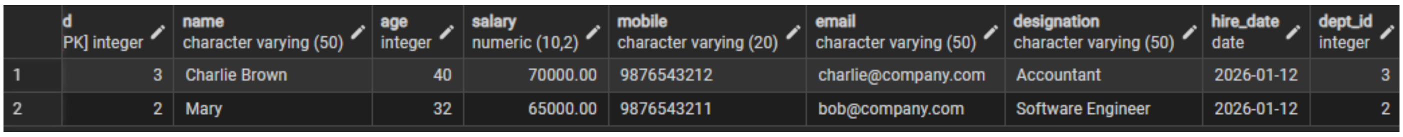
```
NOTICE:  Loop Count: 1
NOTICE:  Loop Count: 2
NOTICE:  Loop Count: 3
NOTICE:  Loop Count: 4
NOTICE:  Loop Count: 5
DO

Query returned successfully in 49 msec.
```

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

For table employee

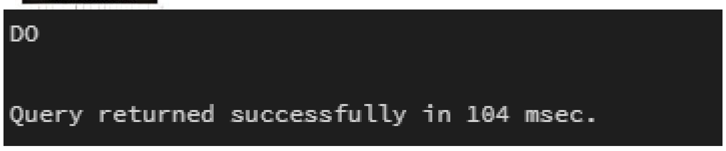


	d PK] integer	name character varying (50)	age integer	salary numeric (10,2)	mobile character varying (20)	email character varying (50)	designation character varying (50)	hire_date date	dept_id integer
1	3	Charlie Brown	40	70000.00	9876543212	charlie@company.com	Accountant	2026-01-12	3
2	2	Mary	32	65000.00	9876543211	bob@company.com	Software Engineer	2026-01-12	2

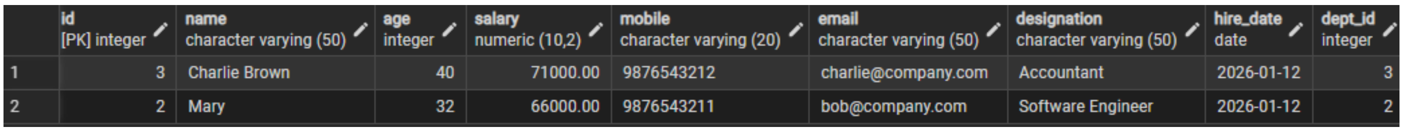
Query:

```
DO $$
DECLARE
   rec RECORD;
BEGIN
   FOR rec IN SELECT id, salary FROM employee LOOP
      UPDATE employee
      SET salary = salary + 1000
      WHERE id = rec.id;
   END LOOP;
END $$;
```

Output:



```
DO

Query returned successfully in 104 msec.
```



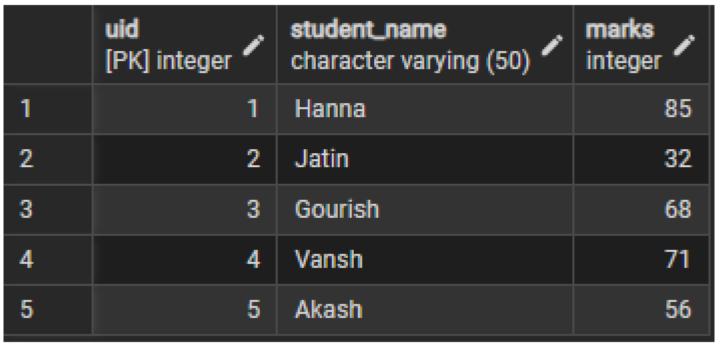
	id [PK] integer	name character varying (50)	age integer	salary numeric (10,2)	mobile character varying (20)	email character varying (50)	designation character varying (50)	hire_date date	dept_id integer
1	3	Charlie Brown	40	71000.00	9876543212	charlie@company.com	Accountant	2026-01-12	3
2	2	Mary	32	66000.00	9876543211	bob@company.com	Software Engineer	2026-01-12	2

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

For table studentgrades



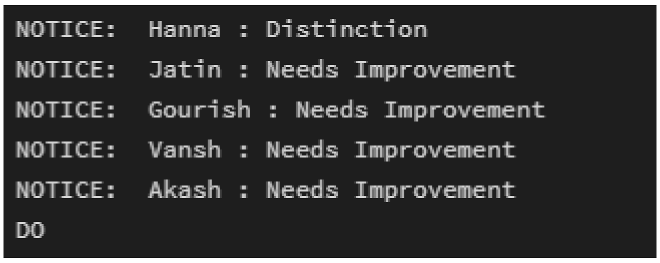
	uid [PK] integer	student_name character varying (50)	marks integer
1	1	Hanna	85
2	2	Jatin	32
3	3	Gourish	68
4	4	Vansh	71
5	5	Akash	56

Query:

```
DO $$
DECLARE
   rec RECORD;
BEGIN
   FOR rec IN SELECT student_name, marks FROM studentgrades LOOP
      IF rec.marks >= 75 THEN
         RAISE NOTICE '% : Distinction', rec.student_name;
      ELSE
         RAISE NOTICE '% : Needs Improvement', rec.student_name;
      END IF;
```

```
    END LOOP;

END $$;
```

Output:



```
NOTICE:  Hanna : Distinction
NOTICE:  Jatin : Needs Improvement
NOTICE:  Gourish : Needs Improvement
NOTICE:  Vansh : Needs Improvement
NOTICE:  Akash : Needs Improvement
DO
```

Learning Outcomes

- Understand the importance of iteration in database programming and procedural SQL.
- Differentiate between FOR, WHILE, and LOOP constructs in PostgreSQL.
- Implement fixed and conditional iterations using PL/pgSQL blocks.
- Perform row-by-row processing of database records using loop structures.
- Apply conditional logic (IF–ELSE) within loops for data classification.
- Relate loop constructs to real-world applications such as payroll processing, reporting, and batch updates.