



**Student Name:** Sahil Hans  
**Branch:** M.C.A(A.I & M.L)  
**Semester:** Second Sem  
**Subject Name:** TECHINCAL SKILLS

**UID:** 25MCI10088  
**Section/Group:** MAM-1 A  
**Date of Performance:** 12/01/2026  
**Subject Code:**

## 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.

**S/W Requirement:** Oracle Database Express Edition and pgAdmin

### **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**

#### **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

Querry:

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

Query returned successfully in 68 msec.
```

### 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 – Violations

	<b>id</b> [PK] integer	<b>entity_name</b> character varying (100)	<b>violation_count</b> integer	<b>approval_status</b> character varying (30)
1	1	Finance_Department	12	Needs Review
2	2	HR_Department	5	Needs Review
3	3	IT_Department	20	Rejected
4	4	Sales_Department	0	Approved
5	5	Admin_Department	0	Approved
6	6	Security_Team	15	Needs Review

Querry:

```
DO $$

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



Output:

```
NOTICE: Entity: Finance_Department, Violations: 12
NOTICE: Entity: HR_Department, Violations: 5
NOTICE: Entity: IT_Department, Violations: 20
NOTICE: Entity: Sales_Department, Violations: 0
NOTICE: Entity: Admin_Department, Violations: 0
NOTICE: Entity: Security_Team, Violations: 15
DO
```

```
Query returned successfully in 68 msec.
```

### 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

Querry:

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

Outcome:

```
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 Employees

	<b>emp_id</b> [PK] integer	<b>fullname</b> character varying (100)	<b>salary</b> numeric (10,2)	<b>dept_id</b> integer	<b>role</b> character varying (50)	<b>email</b> character varying (100)
1	104	Priya Singh	60000.00	3	Accountant	[null]
2	101	Amit Sharma	50000.00	3	HR Executive	[null]

Querry:

DO \$\$

DECLARE

rec RECORD;

BEGIN

FOR rec IN SELECT emp\_id, salary FROM Employees LOOP

    UPDATE Employees

        SET salary = salary - 1000

    WHERE emp\_id = rec.emp\_id;

END LOOP;

END \$\$;

Output:

	<b>emp_id</b> [PK] integer	<b>fullname</b> character varying (100)	<b>salary</b> numeric (10,2)	<b>dept_id</b> integer	<b>role</b> character varying (50)	<b>email</b> character varying (100)
1	104	Priya Singh	61000.00	3	Accountant	[null]
2	101	Amit Sharma	51000.00	3	HR Executive	[null]

### 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

	<b>student_id</b> [PK] integer	<b>student_name</b> character varying (50)	<b>marks</b> integer
1	1	Aarav	95
2	2	Neha	82
3	3	Rohit	68
4	4	Priya	91
5	5	Karan	56
6	6	Simran	45
7	7	Aman	77
8	8	Riya	88
9	9	Vikas	35



Querry:

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 \$\$;

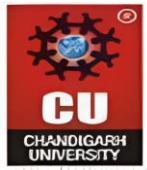
Output:

```
NOTICE: Aarav : Distinction
NOTICE: Neha : Distinction
NOTICE: Rohit : Needs Improvement
NOTICE: Priya : Distinction
NOTICE: Karan : Needs Improvement
NOTICE: Simran : Needs Improvement
NOTICE: Aman : Distinction
NOTICE: Riya : Distinction
NOTICE: Vikas : Needs Improvement
DO

Query returned successfully in 72 msec.
```

## 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.



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

**NAAC**  
**GRADE A+**