UNIVERSITY INSTITUTE of COMPUTING
Asia's Fastest Growing University

NAAC GRADE A+ ACCREDITED UNIVERSITY

Worksheet 4

Student Name:Suyash **UID:25MCI10054**
Branch:MCA (AI&ML) **Section/Group:MAM-1 A**
Semester:2nd **Date of Performance:03/02/2026**
Subject Name:- Technical Training -I **Subject Code: 25CAP-652**

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

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

Experiment Steps

Step 1: FOR Loop – Simple Iteration

```
CREATE TABLE employees (
  emp_id     int PRIMARY KEY,
  emp_name   VARCHAR(50),
  salary     int
);
INSERT INTO employees VALUES
(101, 'Sumit', 45000),
(102, 'Neha', 52000),
(103, 'Rahul', 38000),
(104, 'Priya', 60000),
(105, 'Karan', 48000);


do $$
begin
for i in 1..5 loop
raise notice 'Iteration Number: %', i;
end loop;
end;
$$ ;
```

```
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)

```
do $$
declare
   rec RECORD;
BEGIN
  FOR rec IN (SELECT emp_id, emp_name FROM employees) LOOP
    raise notice 'Employee ID: %, Name: %', rec.emp_id, rec.emp_name;
  END LOOP;
END;
$$;
```

```
NOTICE:  Employee ID: 101, Name: Sumit
NOTICE:  Employee ID: 102, Name: Neha
NOTICE:  Employee ID: 103, Name: Rahul
NOTICE:  Employee ID: 104, Name: Priya
NOTICE:  Employee ID: 105, Name: Karan
DO
```

Example 3: WHILE Loop – Conditional Iteration

```
do $$
declare
   counter INT := 1;
begin
   while counter <= 5 loop
      raise notice 'Counter value: %', counter;
      counter := counter + 1;
   end loop;
end;
$$ ;
```

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

Example 4: LOOP with EXIT WHEN

```
do $$
declare
counter int := 1;
begin
loop
raise notice 'Counter value: %', counter;
counter := counter + 1;
exit when counter > 5;
end loop;
end;
$$ ;
```

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

Example 5: Salary Increment Using FOR Loop

```
do $$
declare
 rec RECORD;
begin
   for rec in
      select emp_id, salary from employees
   loop
      update employees
      set salary = salary * 1.10
      where emp_id = rec.emp_id;

      raise notice 'Updated salary for Employee ID: %', rec.emp_id;
   end loop;
end;
$$ ;
```

```
NOTICE:  Updated salary for Employee ID: 101
NOTICE:  Updated salary for Employee ID: 102
NOTICE:  Updated salary for Employee ID: 103
NOTICE:  Updated salary for Employee ID: 104
NOTICE:  Updated salary for Employee ID: 105
DO
```

Example 6: Combining LOOP with IF Condition

```
do $$
declare
 rec RECORD;
begin
    for rec in
        select emp_id, salary from employees
    loop
if rec.salary > 50000 then
raise notice 'Employee ID % has salary more than 50000', rec.emp_id;
else
raise notice 'Employee ID % has salary less than 50000', rec.emp_id;
end if;
end loop;
end;
$$;
```

```
NOTICE:  Employee ID 101 has salary less than 50000
NOTICE:  Employee ID 102 has salary more than 50000
NOTICE:  Employee ID 103 has salary less than 50000
NOTICE:  Employee ID 104 has salary more than 50000
NOTICE:  Employee ID 105 has salary more than 50000
DO
```

Learning Outcome

- Understand the need for iteration in database applications.
- Identify and use different loop types (FOR, WHILE, LOOP).
- Implement fixed and query-based repetition for row processing.
- Apply conditional and exit-controlled loops for automation tasks.
- Use PL/pgSQL loops in real-world scenarios like payroll, reporting, and batch processing.

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.