

Branch: MCA (Data Science) Kargil	Semester: 2
Student Name: Sameer Shahi	UID: 25MCD10025
Subject Name: Technical Training - I Lab	Subject Code: 25CAP-652
Section/Group: 25MCD KAR-1	Date of Performance: 03-02-2026

EXPERIMENT – 04

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
-

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 (Conceptual Explanation)

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

Queries:

```
do $$  
declare  
  i integer;  
begin  
  for i in 1..5 loop  
    raise notice 'iteration number: %', i;  
  end loop;  
end $$;
```

Output:

Data Output	Messages	Notifications
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 189 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

Queries:

```
create table employee (
    emp_id serial primary key,
    emp_name varchar(50),
    salary numeric(10,2)
);

insert into employee (emp_name, salary) values
('arjun', 45000),
('meera', 72000),
('rohan', 88000),
('kavya', 60000),
('sameer', 95000);

do $$
declare
    rec record;
begin
    for rec in
        select emp_id, emp_name from employee order by emp_id
    loop
        raise notice 'emp id: %, name: %', rec.emp_id, rec.emp_name;
    end loop;
end $$;
```

Output:

Data Output	Messages	Notifications
NOTICE:	emp id: 1, name: arjun	
NOTICE:	emp id: 2, name: meera	
NOTICE:	emp id: 3, name: rohan	
NOTICE:	emp id: 4, name: kavya	
NOTICE:	emp id: 5, name: sameer	
DO		
Query returned successfully in 77 msec.		

Example 3: WHILE Loop – Conditional Iteration

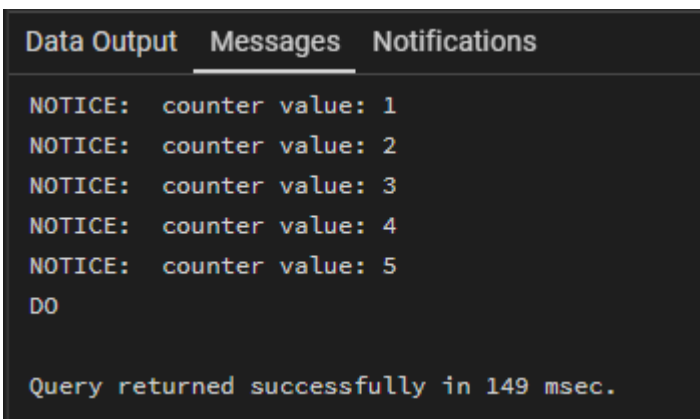
- The loop runs until a condition becomes false
- Execution depends entirely on the condition
- The condition is checked before every iteration

Application: Retry mechanisms, validation loops

Queries:

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

Output:



```
Data Output  Messages  Notifications  
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 149 msec.
```

Example 4: LOOP with EXIT WHEN

- The loop does not stop automatically

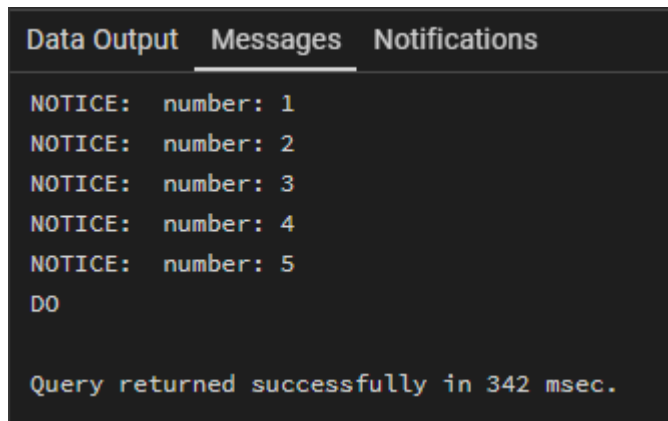
- An explicit exit condition controls termination
- Gives flexibility in complex logic

Application: Workflow engines, complex decision cycles

Queries:

```
do $$  
declare  
    num integer := 1;  
begin  
    loop  
        raise notice 'number: %', num;  
        num := num + 1;  
  
        exit when num > 5;  
    end loop;  
end $$;
```

Output:



```
Data Output  Messages  Notifications  
NOTICE:  number: 1  
NOTICE:  number: 2  
NOTICE:  number: 3  
NOTICE:  number: 4  
NOTICE:  number: 5  
DO  
  
Query returned successfully in 342 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

Queries:

```
update employee set salary =
case emp_name
when 'arjun' then 45000
when 'meera' then 72000
when 'rohan' then 88000
when 'kavya' then 60000
when 'sameer' then 95000
end;

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 emp id: %', rec.emp_id;
    end loop;
end $$;
```

Output:

Data Output	Messages	Notifications
NOTICE:	salary incremented for emp id: 1	
NOTICE:	salary incremented for emp id: 2	
NOTICE:	salary incremented for emp id: 3	
NOTICE:	salary incremented for emp id: 4	
NOTICE:	salary incremented for emp id: 5	
DO		
Query returned successfully in 86 msec.		

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

Queries:

```
do $$  
declare  
    rec record;  
begin  
    for rec in  
        select emp_id, salary from employee  
    loop  
        if rec.salary >= 80000 then  
            raise notice 'emp id % classified as high income', rec.emp_id;  
        else  
            raise notice 'emp id % classified as regular income', rec.emp_id;  
        end if;  
    end loop;  
end $$;
```

Output:

Data Output	Messages	Notifications
NOTICE:	emp id 1	classified as regular income
NOTICE:	emp id 2	classified as regular income
NOTICE:	emp id 3	classified as high income
NOTICE:	emp id 4	classified as regular income
NOTICE:	emp id 5	classified as high income
DO		
Query returned successfully in 115 msec.		

Learning outcomes (What I have learnt)

- Understood the need for iteration in database programming
- Learnt the working of FOR, WHILE, and LOOP constructs in PostgreSQL
- Understood how repeated data processing is handled using PL/pgSQL
- Learnt how loops are applied in real-world database scenarios
- Gained foundational knowledge of procedural SQL used in enterprise systems