UNIVERSITY INSTITUTE of COMPUTING
Asia's Fastest Growing University

NAAC GRADE A+ ACCREDITED UNIVERSITY

Worksheet-4

Name: Aman Kumar
Branch: MCA General
Semester: II
Subject Name: Technical Training

UID: 25MCA20128
Section/Group: 25MCA_KAR-1
Date of Performance: 03-02-26
Subject Code: 25CAP-652

Aim of the Session:

The aim of this practical session is to understand and implement iterative control structures in PostgreSQL using FOR, WHILE, and LOOP constructs. This practical helps in learning how repetitive tasks are handled in database programming and how procedural SQL is used to process data repeatedly in real-world database applications such as payroll processing, reporting, and batch operations.

Objective of the Session:

The objectives of this practical session are:

- To understand why iteration is required in database programming
- To learn the syntax and working of FOR, WHILE, and LOOP structures
- To analyze how repeated execution of SQL statements is achieved
- To relate loop constructs with real-world database operations
- To gain conceptual clarity of PL/pgSQL used in enterprise systems

Practical / Experiment Steps:

Step 1: Understanding the Need for Iteration

- Identify situations where a task must be repeated multiple times, such as processing multiple records or executing repeated logic.

Step 2: Implementing FOR Loop (Simple Iteration)

- Use a FOR loop to execute a block of code a fixed number of times.

Step 3: Implementing FOR Loop with Query

- Process database records row by row using a FOR loop based on query results.

Step 4: Implementing WHILE Loop

- Execute a block of code repeatedly based on a condition that is checked before each iteration.

Step 5: Implementing LOOP with EXIT Condition

- Create an infinite loop and control its termination using an explicit EXIT condition.

Step 6: Applying Iteration in Real-World Scenario

- Use loops to simulate salary increment and conditional processing of employee records.

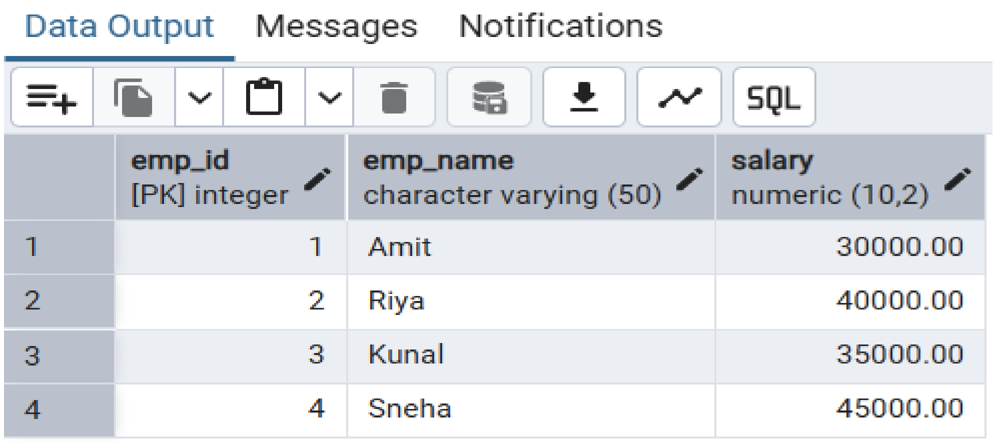
Procedure of the Practical:

1. Start the system and log in to the computer.
2. Open the PostgreSQL client tool (psql / pgAdmin).
3. Create a table to store employee details.

```
CREATE TABLE employees (
    emp_id SERIAL PRIMARY KEY,
    emp_name VARCHAR(50),
    salary NUMERIC(10,2)
);
```

4. Insert sample records into the table.

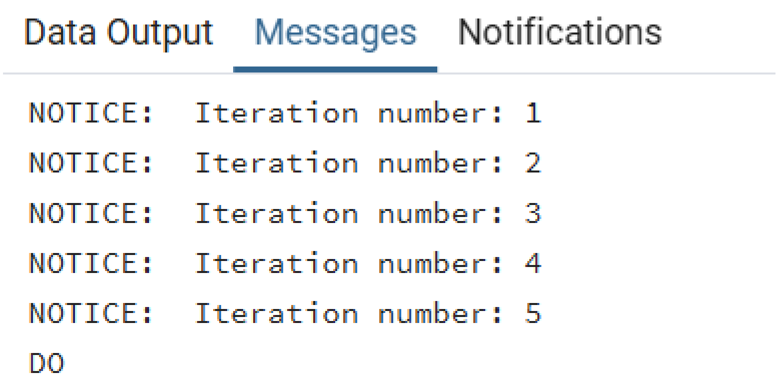
```
INSERT INTO employees (emp_name, salary) VALUES
('Amit', 30000),
('Riya', 40000),
('Kunal', 35000),
('Sneha', 45000);
```



	emp_id [PK] integer	emp_name character varying (50)	salary numeric (10,2)
1	1	Amit	30000.00
2	2	Riya	40000.00
3	3	Kunal	35000.00
4	4	Sneha	45000.00

5. FOR Loop – Simple Iteration.

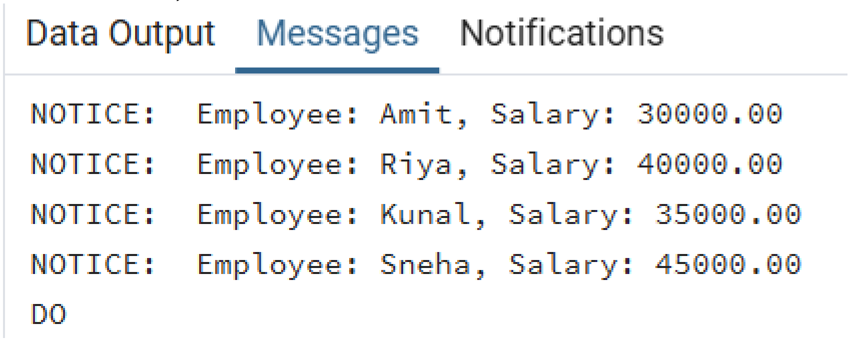
```
DO $$
BEGIN
   FOR i IN 1..5 LOOP
      RAISE NOTICE 'Iteration number: %', i;
   END LOOP;
END $$;
```



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

6. FOR Loop with Query (Row-by-Row Processing).

```
DO $$
DECLARE
   rec RECORD;
BEGIN
   FOR rec IN SELECT emp_name, salary FROM employees LOOP
      RAISE NOTICE 'Employee: %, Salary: %', rec.emp_name, rec.salary;
   END LOOP;
END $$;
```

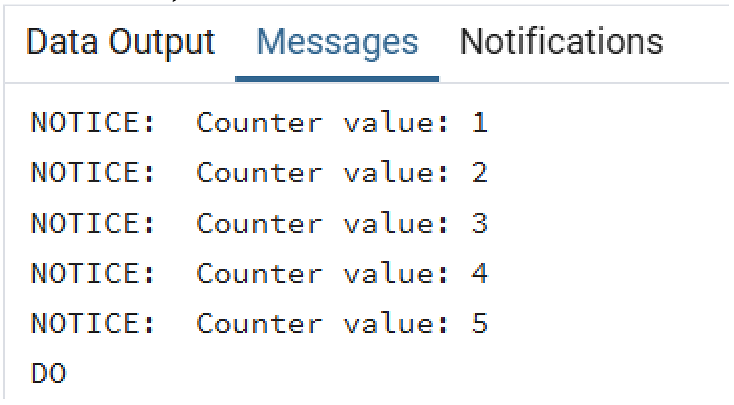


```
NOTICE:  Employee: Amit, Salary: 30000.00
NOTICE:  Employee: Riya, Salary: 40000.00
NOTICE:  Employee: Kunal, Salary: 35000.00
NOTICE:  Employee: Sneha, Salary: 45000.00
DO
```

7. WHILE Loop – Conditional Iteration.

```
DO $$
DECLARE
   counter INT := 1;
BEGIN
```

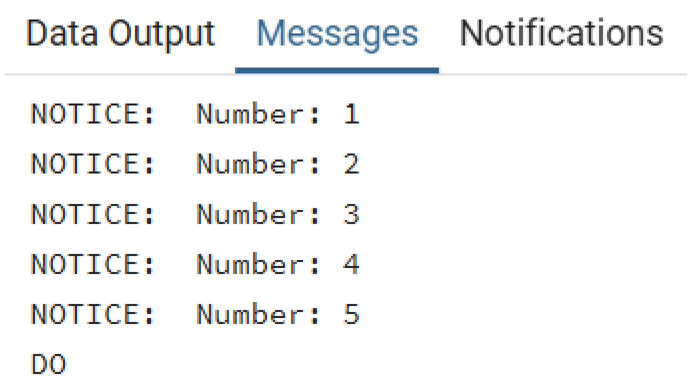
```
    WHILE counter <= 5 LOOP
        RAISE NOTICE 'Counter value: %', counter;
        counter := counter + 1;
    END LOOP;
END $$;
```



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

8. LOOP with EXIT WHEN Condition.

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

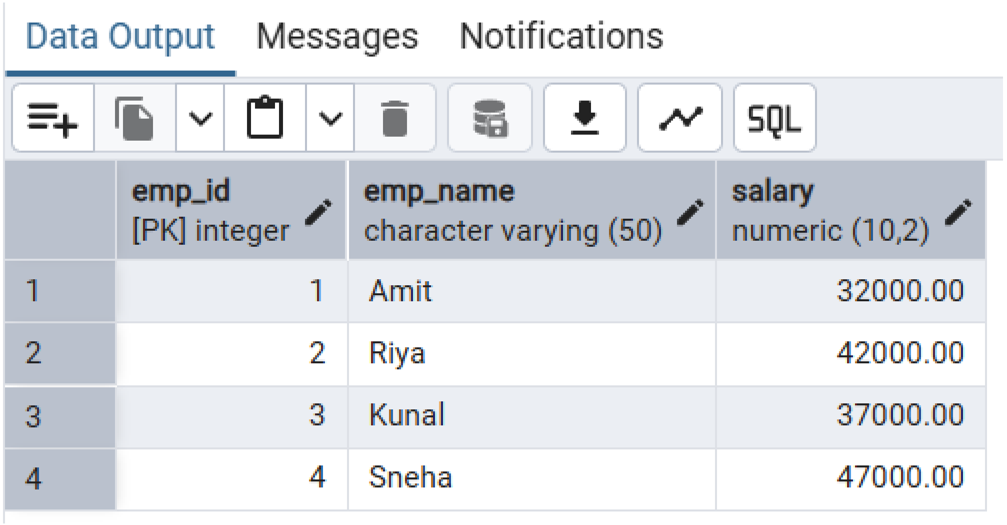


```
NOTICE:  Number: 1
NOTICE:  Number: 2
NOTICE:  Number: 3
NOTICE:  Number: 4
NOTICE:  Number: 5
DO
```

9. Salary Increment Using FOR Loop.

```
DO $$
DECLARE
    rec RECORD;
BEGIN
```

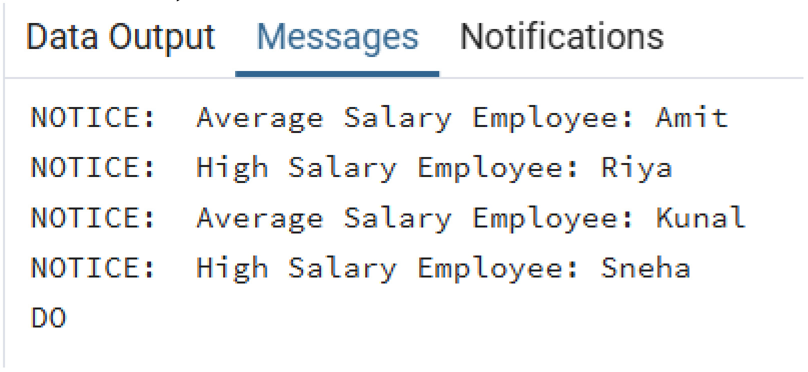
```
    FOR rec IN SELECT emp_id, salary FROM employees LOOP
        UPDATE employees
        SET salary = salary + 2000
        WHERE emp_id = rec.emp_id;
    END LOOP;
END $$;
```



	emp_id [PK] integer	emp_name character varying (50)	salary numeric (10,2)
1	1	Amit	32000.00
2	2	Riya	42000.00
3	3	Kunal	37000.00
4	4	Sneha	47000.00

10. Combining LOOP with IF Condition.

```
DO $$
DECLARE
    rec RECORD;
BEGIN
    FOR rec IN SELECT emp_name, salary FROM employees LOOP
        IF rec.salary >= 40000 THEN
            RAISE NOTICE 'High Salary Employee: %', rec.emp_name;
        ELSE
            RAISE NOTICE 'Average Salary Employee: %', rec.emp_name;
        END IF;
    END LOOP;
END $$;
```



```
NOTICE:  Average Salary Employee: Amit
NOTICE:  High Salary Employee: Riya
NOTICE:  Average Salary Employee: Kunal
NOTICE:  High Salary Employee: Sneha
DO
```

11. Verify the output after execution.
12. Note down the results obtained.
13. Save the work and take screenshots for record.

I/O Analysis:

Input Provided

- PL/pgSQL blocks using FOR, WHILE, and LOOP
- Sample employee records
- Conditions and loop counters

Output Generated

- Iterative messages showing loop execution
- Employee details processed row by row
- Updated salary values after iteration
- Conditional classification of employee data

Learning Outcomes:

- Understanding why iteration is important in database programming
- Learning how FOR, WHILE, and LOOP structures work in PostgreSQL
- Gaining knowledge of row-by-row data processing
- Applying loops in real-world scenarios such as payroll systems
- Building a strong foundation in procedural SQL for enterprise applications