

Experiment No. 4

Student Name: Sanchit Katoch

Branch: MCA-General

Semester: 2nd

Subject Name: Technical Training-1

UID: 25MCA20059

Section/Group: 25MCA-1-A

Date of Performance: 03/02/26

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.

Objective:

- To understand why iteration is required in database programming.
- To learn the purpose and behaviour 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.

Tools Used:

PostgreSQL

Procedure:

Step 1: FOR Loop - Simple Iteration

- The loop runs a fixed number of times

- Each iteration represents one execution cycle
- Useful for understanding basic loop behaviour

Step 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

Step 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

Step 4: LOOP with EXIT WHEN

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

Step 5: Salary Increment Using FOR Loop

- Employee records are processed one by one
- Salary values are updated iteratively
- Represents real-world payroll processing

Step 6: Combining LOOP with IF Condition

- Loop processes each record
- Conditional logic classifies data during iteration
- Demonstrates decision-making inside loops

Code:

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

```
END LOOP;
END $$;
--2
CREATE TABLE employees (
    emp_id INT,
    emp_name VARCHAR(50),
    salary INT
);

INSERT INTO employees VALUES
(1, 'Amit', 30000),
(2, 'Neha', 45000),
(3, 'Rahul', 28000);

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

--3
DO $$
DECLARE
    counter INT := 1;
BEGIN
    WHILE counter <= 5 LOOP
        RAISE NOTICE 'Counter value: %', counter;
        counter := counter + 1;
    END LOOP;
END $$;

--4
DO $$
DECLARE
    num INT := 1;
BEGIN
    LOOP
```

```
RAISE NOTICE 'Number: %', num;  
num := num + 1;
```

```
EXIT WHEN num > 5;  
END LOOP;  
END $$;
```

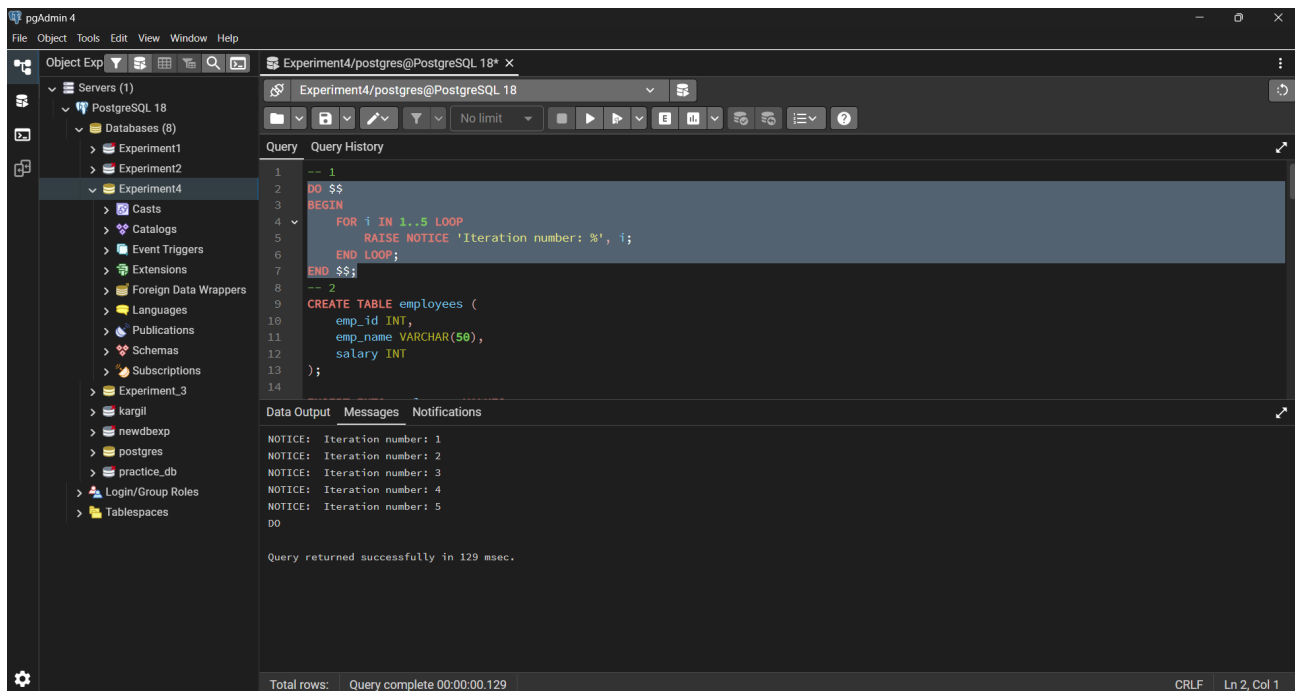
```
--5  
DO $$  
DECLARE  
    rec RECORD;  
BEGIN  
    FOR rec IN SELECT * FROM employees LOOP  
        UPDATE employees  
        SET salary = salary + 5000  
        WHERE emp_id = rec.emp_id;  
    END LOOP;  
END $$;
```

```
SELECT * FROM employees;
```

```
-- 6  
DO $$  
DECLARE  
    rec RECORD;  
BEGIN  
    FOR rec IN SELECT * FROM employees LOOP  
        IF rec.salary >= 40000 THEN  
            RAISE NOTICE '% is High Salary Employee', rec.emp_name;  
        ELSE  
            RAISE NOTICE '% is Low Salary Employee', rec.emp_name;  
        END IF;  
    END LOOP;  
END $$;
```

Output:

Step1: FOR Loop - Simple Iteration



The screenshot shows the pgAdmin 4 interface. The left pane displays the database structure for 'PostgreSQL 18', with 'Experiment4' selected. The main pane shows a SQL query with a FOR loop. The 'Data Output' tab is active, displaying the results of the loop iterations.

```
1  -- 1
2  DO $$
3  BEGIN
4      FOR i IN 1..5 LOOP
5          RAISE NOTICE 'Iteration number: %', i;
6      END LOOP;
7  END $$;
8  -- 2
9  CREATE TABLE employees (
10     emp_id INT,
11     emp_name VARCHAR(50),
12     salary INT
13 );
```

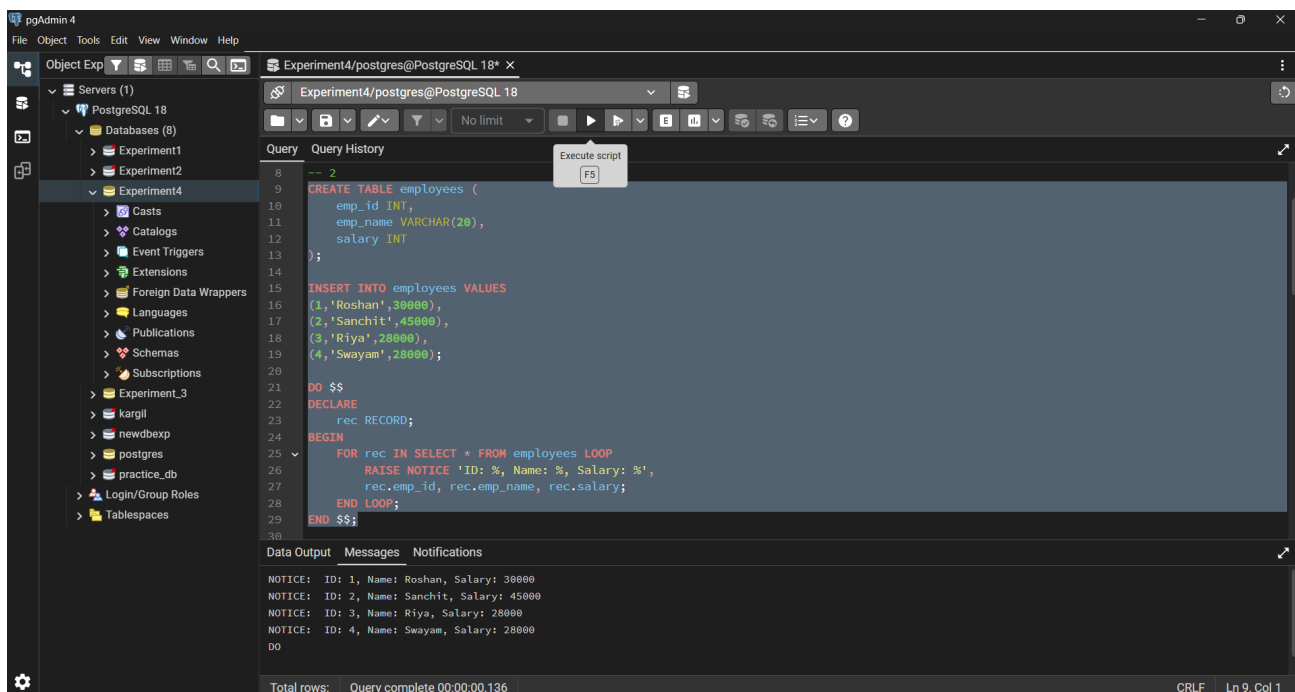
Data Output Messages Notifications

```
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 129 msec.

Total rows: Query complete 00:00:00.129 CRLF Ln 2, Col 1

Step2: FOR Loop with Query (Row-by-Row Processing)



The screenshot shows the pgAdmin 4 interface. The left pane displays the database structure for 'PostgreSQL 18', with 'Experiment4' selected. The main pane shows a SQL query that creates a table, inserts data, and then uses a FOR loop to process each row. The 'Data Output' tab is active, displaying the results of the loop iterations.

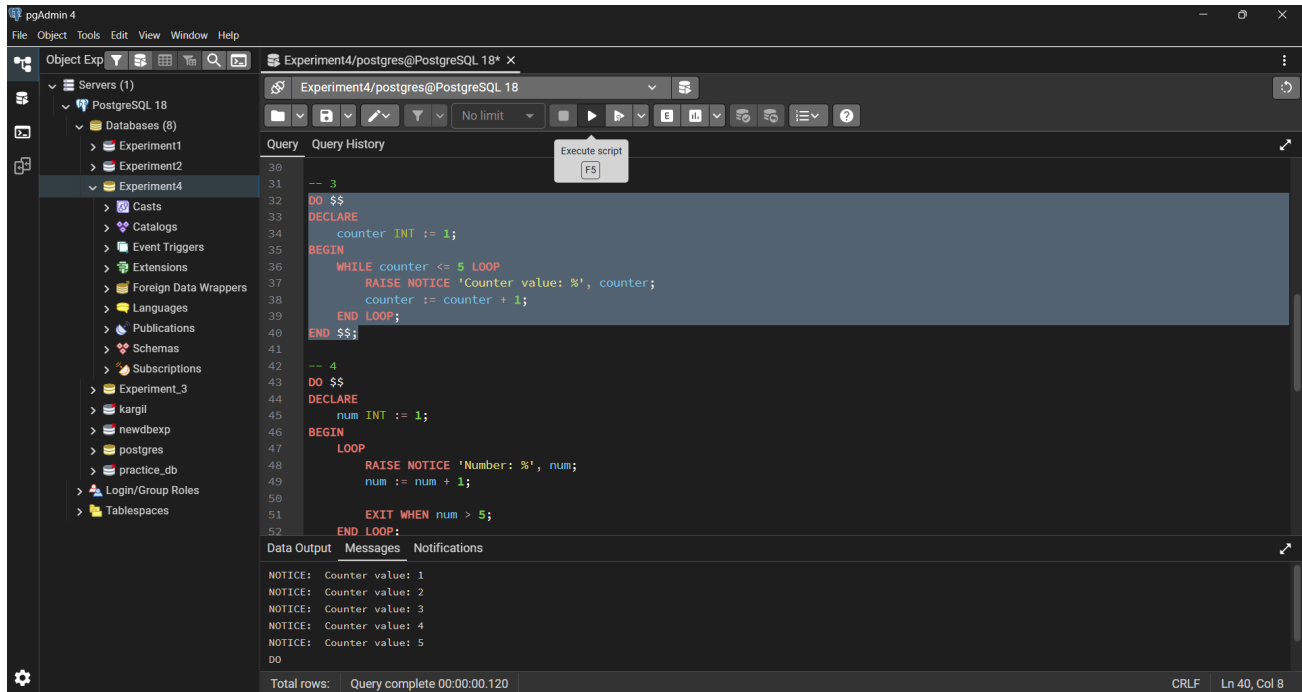
```
8  -- 2
9  CREATE TABLE employees (
10     emp_id INT,
11     emp_name VARCHAR(20),
12     salary INT
13 );
14
15 INSERT INTO employees VALUES
16 (1,'Roshan',30000),
17 (2,'Sanchit',45000),
18 (3,'Riya',28000),
19 (4,'Swayam',28000);
20
21 DO $$
22 DECLARE
23     rec RECORD;
24 BEGIN
25     FOR rec IN SELECT * FROM employees LOOP
26         RAISE NOTICE 'ID: %, Name: %, Salary: %',
27             rec.emp_id, rec.emp_name, rec.salary;
28     END LOOP;
29 END $$;
```

Data Output Messages Notifications

```
NOTICE: ID: 1, Name: Roshan, Salary: 30000
NOTICE: ID: 2, Name: Sanchit, Salary: 45000
NOTICE: ID: 3, Name: Riya, Salary: 28000
NOTICE: ID: 4, Name: Swayam, Salary: 28000
DO
```

Total rows: Query complete 00:00:00.136 CRLF Ln 9, Col 1

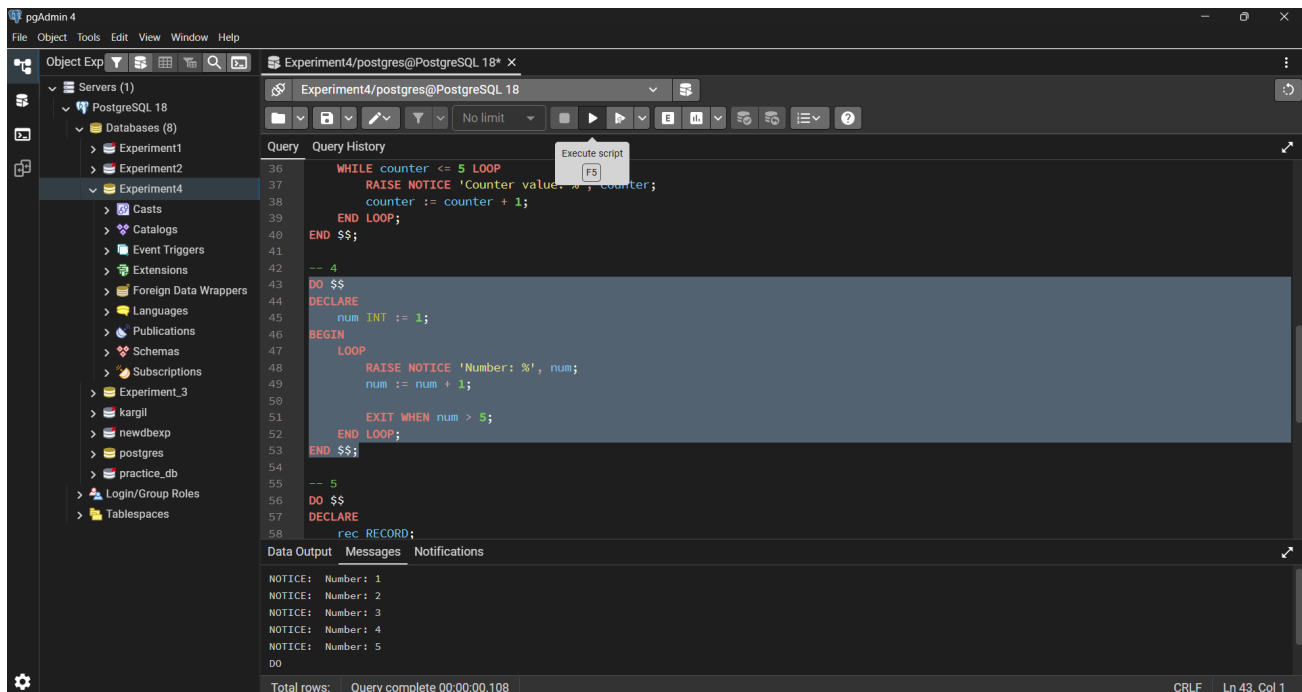
Step3: WHILE Loop - Conditional Iteration



The screenshot shows the pgAdmin 4 interface with a query editor containing a SQL script. The script defines two variables, 'counter' and 'num', and uses a WHILE loop to iterate from 1 to 5, printing the counter value. The output pane shows the results of the loop.

```
30
31 -- 3
32 DO $$
33 DECLARE
34     counter INT := 1;
35 BEGIN
36     WHILE counter <= 5 LOOP
37         RAISE NOTICE 'Counter value: %', counter;
38         counter := counter + 1;
39     END LOOP;
40 END $$;
41
42 -- 4
43 DO $$
44 DECLARE
45     num INT := 1;
46 BEGIN
47     LOOP
48         RAISE NOTICE 'Number: %', num;
49         num := num + 1;
50     END LOOP;
51     EXIT WHEN num > 5;
52 END LOOP;
53
54 Data Output Messages Notifications
55 NOTICE: Counter value: 1
56 NOTICE: Counter value: 2
57 NOTICE: Counter value: 3
58 NOTICE: Counter value: 4
59 NOTICE: Counter value: 5
60 DO
61 Total rows: Query complete 00:00:00.120 CRLF Ln 40, Col 8
```

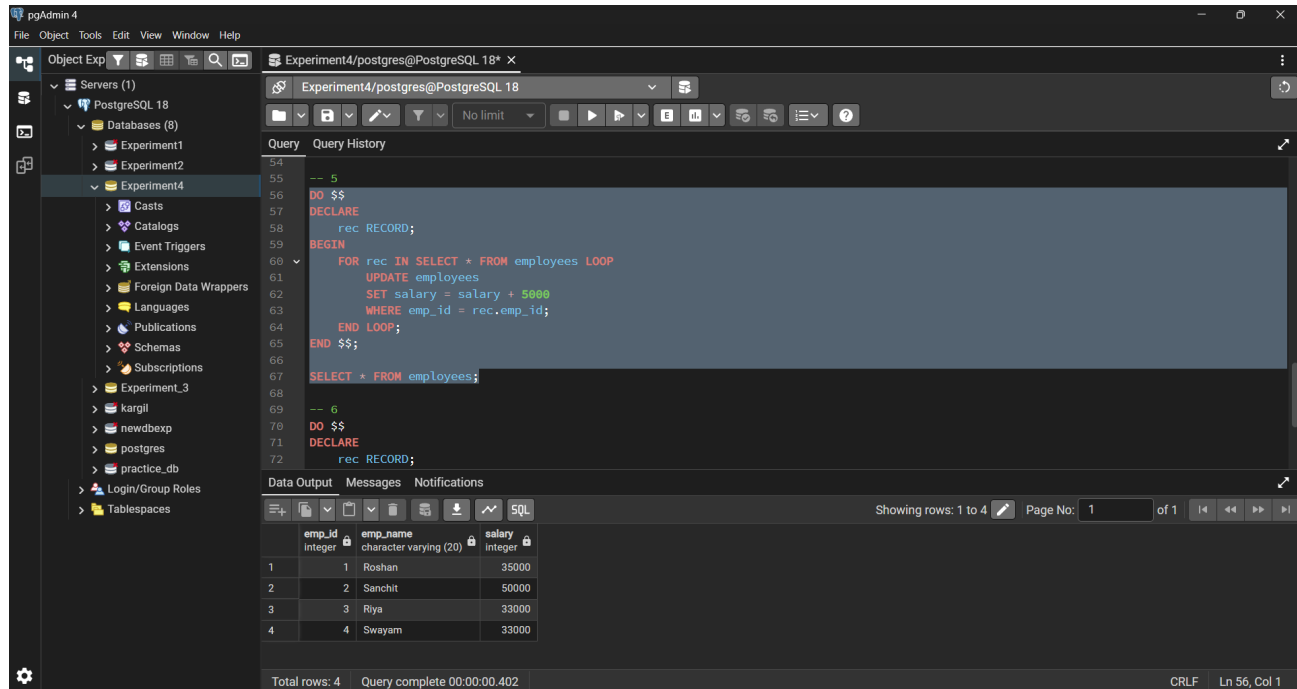
Step4: LOOP with EXIT WHEN



The screenshot shows the pgAdmin 4 interface with a query editor containing a SQL script. The script defines a variable 'num' and uses a LOOP with an EXIT WHEN clause to iterate from 1 to 5, printing the number. The output pane shows the results of the loop.

```
36 WHILE counter <= 5 LOOP
37     RAISE NOTICE 'Counter value: %', counter;
38     counter := counter + 1;
39 END LOOP;
40 END $$;
41
42 -- 4
43 DO $$
44 DECLARE
45     num INT := 1;
46 BEGIN
47     LOOP
48         RAISE NOTICE 'Number: %', num;
49         num := num + 1;
50     END LOOP;
51     EXIT WHEN num > 5;
52 END LOOP;
53 END $$;
54
55 -- 5
56 DO $$
57 DECLARE
58     rec RECORD;
59 Data Output Messages Notifications
60 NOTICE: Number: 1
61 NOTICE: Number: 2
62 NOTICE: Number: 3
63 NOTICE: Number: 4
64 NOTICE: Number: 5
65 DO
66 Total rows: Query complete 00:00:00.108 CRLF Ln 43, Col 1
```

Step5: Salary Increment Using FOR Loop



The screenshot shows the pgAdmin 4 interface with a PostgreSQL 18 database. The left sidebar shows the database structure, including a table named 'employees' under the 'Experiment4' schema. The main query editor displays the following SQL code:

```

54
55 -- 5
56 DO $$
57 DECLARE
58   rec RECORD;
59 BEGIN
60   FOR rec IN SELECT * FROM employees LOOP
61     UPDATE employees
62     SET salary = salary + 5000
63     WHERE emp_id = rec.emp_id;
64   END LOOP;
65 END $$;
66
67 SELECT * FROM employees;
68
69 -- 6
70 DO $$
71 DECLARE
72   rec RECORD;

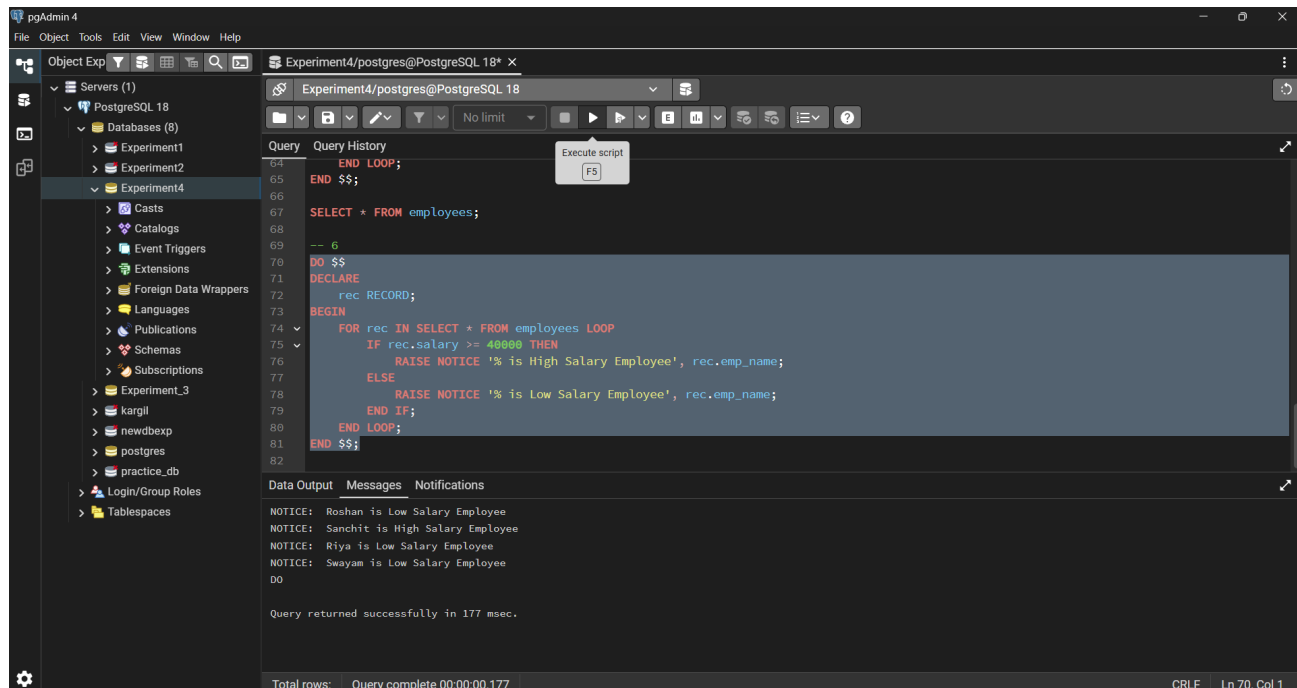
```

The 'Data Output' tab shows the result of the query, displaying the 'employees' table with 4 rows:

emp_id	emp_name	salary
1	Roshan	35000
2	Sanchit	50000
3	Riya	33000
4	Swayam	33000

Total rows: 4 Query complete 00:00:00.402

Step6: Combining LOOP with IF Condition



The screenshot shows the pgAdmin 4 interface with a PostgreSQL 18 database. The left sidebar shows the database structure, including a table named 'employees' under the 'Experiment4' schema. The main query editor displays the following SQL code:

```

64   END LOOP;
65 END $$;
66
67 SELECT * FROM employees;
68
69 -- 6
70 DO $$
71 DECLARE
72   rec RECORD;
73 BEGIN
74   FOR rec IN SELECT * FROM employees LOOP
75     IF rec.salary >= 40000 THEN
76       RAISE NOTICE '% is High Salary Employee', rec.emp_name;
77     ELSE
78       RAISE NOTICE '% is Low Salary Employee', rec.emp_name;
79     END IF;
80   END LOOP;
81 END $$;
82

```

The 'Data Output' tab shows the result of the query, displaying the 'employees' table with 4 rows. The output also includes notices for each employee's salary status:

```

NOTICE: Roshan is Low Salary Employee
NOTICE: Sanchit is High Salary Employee
NOTICE: Riya is Low Salary Employee
NOTICE: Swayam is Low Salary Employee
DO

```

Query returned successfully in 177 msec.

Total rows: Query complete 00:00:00.177

Learning Outcomes:

- Understood the importance of iteration in database programming for repeated execution of logic.
- Learnt the working and use of FOR, WHILE, and LOOP constructs in PostgreSQL.
- Gained practical knowledge of row-by-row data processing using loops in PL/SQL.
- Understood how iterative updates and conditional logic can be applied inside loops.