

## Experiment No. 5

**Student Name:** Roshan Kumar Singh

**Branch:** MCA(GEN)

**Semester:** 2<sup>nd</sup>

**Subject Name:** Technical Training-1

**UID:** 25MCA20067

**Section/Group:** 25MCA-1\_A

**Date of Performance:** 24/02/26

**Subject Code:** 25CAP-652

### Aim:

To gain hands-on experience in creating and using cursors in PostgreSQL for row-by-row data processing, enabling sequential access and manipulation of query results.

### Objective:

- To understand the concept and need of cursors in PostgreSQL
- To learn sequential data access using cursors
- To perform row-level manipulation using cursor logic
- To understand cursor lifecycle management
- To handle cursor termination and exceptions.

### Tools Used:

PostgreSQL

### Procedure:

Step 1: Implementing a Simple Forward-Only Cursor

- Creating a cursor to loop through an employee table and print individual records.

## Step 2: Complex Row-by-Row Manipulation

- Using a cursor to update salaries based on a dynamic "Experience-to-Performance" ratio logic.

## Step 3: Exception and Status Handling

- Ensuring the cursor handles empty result sets or termination signals gracefully.

### Code:

```
CREATE TABLE employee (  
    emp_id INT PRIMARY KEY,  
    emp_name VARCHAR(50),  
    salary NUMERIC,  
    experience INT,  
    performance VARCHAR(1)  
);
```

```
INSERT INTO employee VALUES  
(1, 'Roshan', 25000, 5, 'B'),  
(2, 'Swayam', 40000, 3, 'A'),  
(3, 'Sanchit', 25000, 2, 'C'),  
(4, 'Ankush', 30000, 4, 'A'),  
(5, 'Riya', 30000, 3, 'B');
```

```
--1  
DO $$  
DECLARE  
    emp_cursor CURSOR FOR  
        SELECT emp_id, emp_name, salary FROM employee;  
    rec RECORD;  
BEGIN  
    OPEN emp_cursor;  
  
    LOOP  
        FETCH emp_cursor INTO rec;  
        EXIT WHEN NOT FOUND;  
        RAISE NOTICE 'ID: %, Name: %, Salary: %',
```

```
rec.emp_id, rec.emp_name, rec.salary;  
END LOOP;
```

```
CLOSE emp_cursor;  
END $$;
```

```
--2
```

```
DO $$
```

```
DECLARE
```

```
emp_cursor CURSOR FOR
```

```
SELECT emp_id, salary, experience, performance FROM employee;
```

```
rec RECORD;
```

```
new_salary NUMERIC;
```

```
BEGIN
```

```
OPEN emp_cursor;
```

```
LOOP
```

```
FETCH emp_cursor INTO rec;
```

```
EXIT WHEN NOT FOUND;
```

```
IF rec.experience >= 5 AND rec.performance = 'A' THEN
```

```
new_salary := rec.salary * 1.20;
```

```
ELSIF rec.experience >= 3 AND rec.performance = 'B' THEN
```

```
new_salary := rec.salary * 1.10;
```

```
ELSE
```

```
new_salary := rec.salary * 1.05;
```

```
END IF;
```

```
UPDATE employee
```

```
SET salary = new_salary
```

```
WHERE emp_id = rec.emp_id;
```

```
END LOOP;
```

```
CLOSE emp_cursor;
```

```
END $$;
```

```
--3
```

```
DO $$
```

```
DECLARE
```

```
emp_cursor CURSOR FOR SELECT * FROM employee;
```

```
rec RECORD;
```

```
BEGIN
```

```
OPEN emp_cursor;
```

```
LOOP
```

```
    FETCH emp_cursor INTO rec;
```

```
    EXIT WHEN NOT FOUND;
```

```
    RAISE NOTICE 'Processing Employee: %', rec.emp_name;
```

```
END LOOP;
```

```
CLOSE emp_cursor;
```

```
EXCEPTION
```

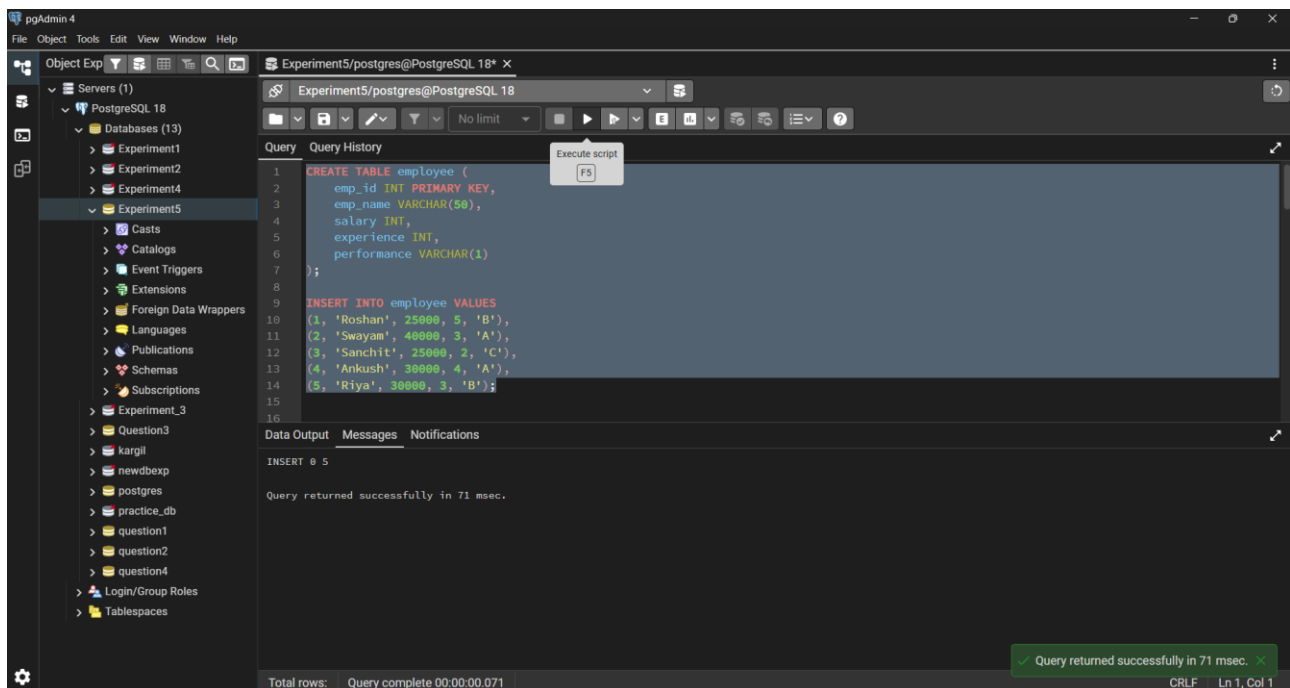
```
    WHEN OTHERS THEN
```

```
        RAISE NOTICE 'Error occurred: %', SQLERRM;
```

```
END $$;
```

## Output:

Table create and data insert



The screenshot displays the pgAdmin 4 application window. The left sidebar shows a tree view of the database structure, with 'Experiment5' selected under 'Databases (13)'. The main pane shows the 'Query' editor with the following SQL script:

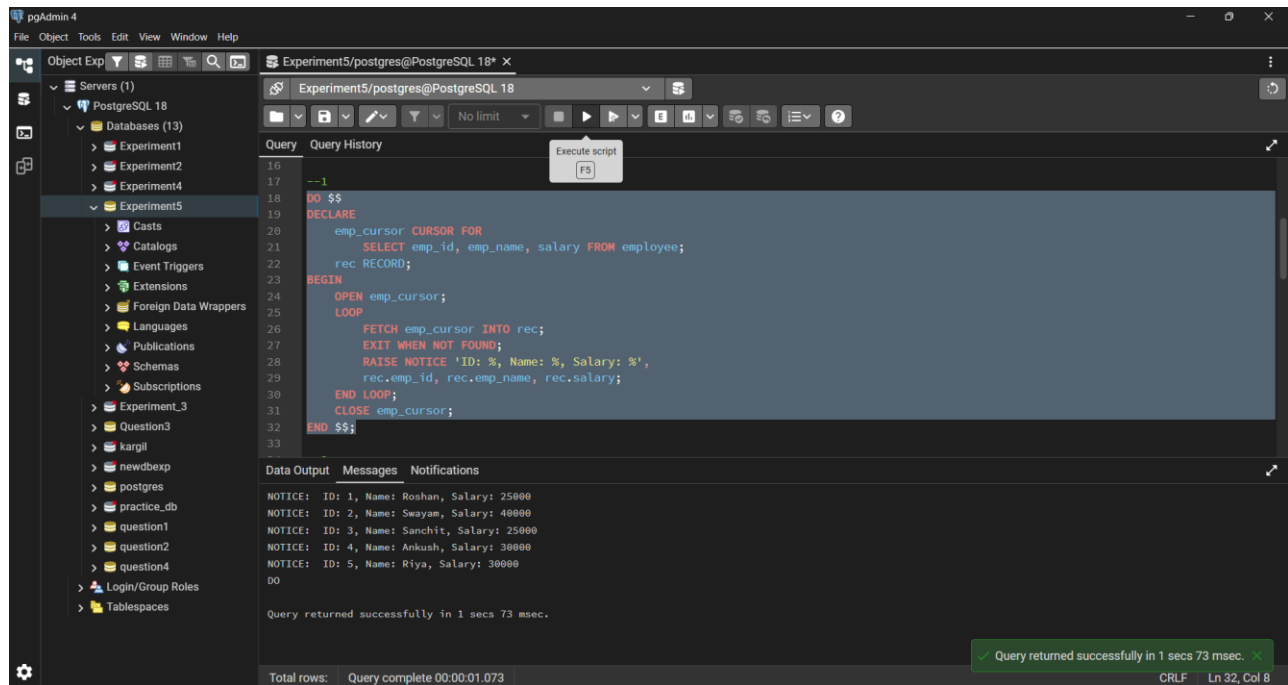
```
1 CREATE TABLE employee (  
2     emp_id INT PRIMARY KEY,  
3     emp_name VARCHAR(50),  
4     salary INT,  
5     experience INT,  
6     performance VARCHAR(1)  
7 );  
8  
9 INSERT INTO employee VALUES  
10 (1, 'Roshan', 25000, 5, 'B'),  
11 (2, 'Swayam', 40000, 3, 'A'),  
12 (3, 'Sanchit', 25000, 2, 'C'),  
13 (4, 'Ankush', 30000, 4, 'A'),  
14 (5, 'Riya', 30000, 3, 'B');
```

The 'Data Output' tab is active, showing the result of the INSERT statement:

```
INSERT 0 5  
  
Query returned successfully in 71 msec.
```

A green status bar at the bottom right indicates 'Query returned successfully in 71 msec.' and 'CRLF Ln 1, Col 1'.

## Step1: Implementing a Simple Forward-Only Cursor



The screenshot shows the pgAdmin 4 interface with a PostgreSQL 18 database. The left sidebar shows the database structure, including a schema named 'Experiment5'. The main query editor displays the following SQL code:

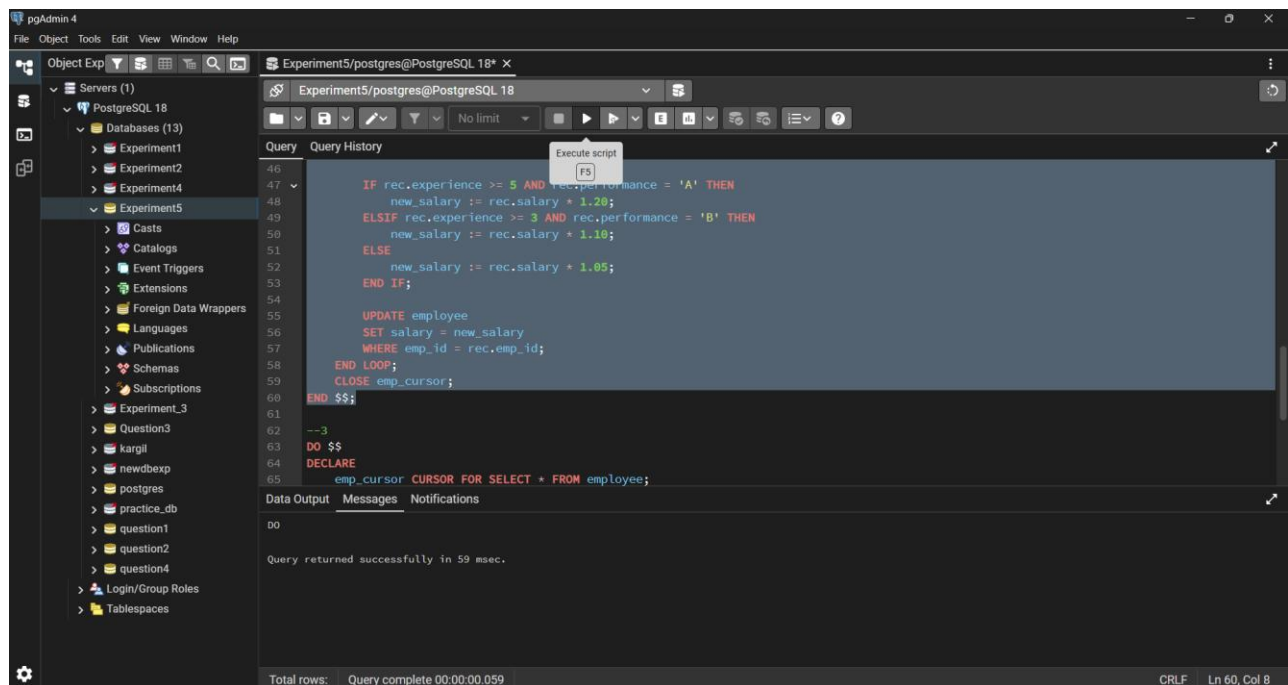
```
--1
DO $$
DECLARE
    emp_cursor CURSOR FOR
        SELECT emp_id, emp_name, salary FROM employee;
    rec RECORD;
BEGIN
    OPEN emp_cursor;
    LOOP
        FETCH emp_cursor INTO rec;
        EXIT WHEN NOT FOUND;
        RAISE NOTICE 'ID: %, Name: %, Salary: %',
            rec.emp_id, rec.emp_name, rec.salary;
    END LOOP;
    CLOSE emp_cursor;
END $$;
```

The 'Data Output' tab shows the results of the query, displaying five rows of employee data as notices:

```
NOTICE: ID: 1, Name: Roshan, Salary: 25000
NOTICE: ID: 2, Name: Swayam, Salary: 40000
NOTICE: ID: 3, Name: Sanchit, Salary: 25000
NOTICE: ID: 4, Name: Ankush, Salary: 30000
NOTICE: ID: 5, Name: Riya, Salary: 30000
DO
```

The status bar at the bottom indicates: 'Query returned successfully in 1 secs 73 msec.' and 'Total rows: Query complete 00:00:01.073'.

## Step2: Complex Row-by-Row Manipulation



The screenshot shows the pgAdmin 4 interface with a PostgreSQL 18 database. The left sidebar shows the database structure, including a schema named 'Experiment5'. The main query editor displays the following SQL code:

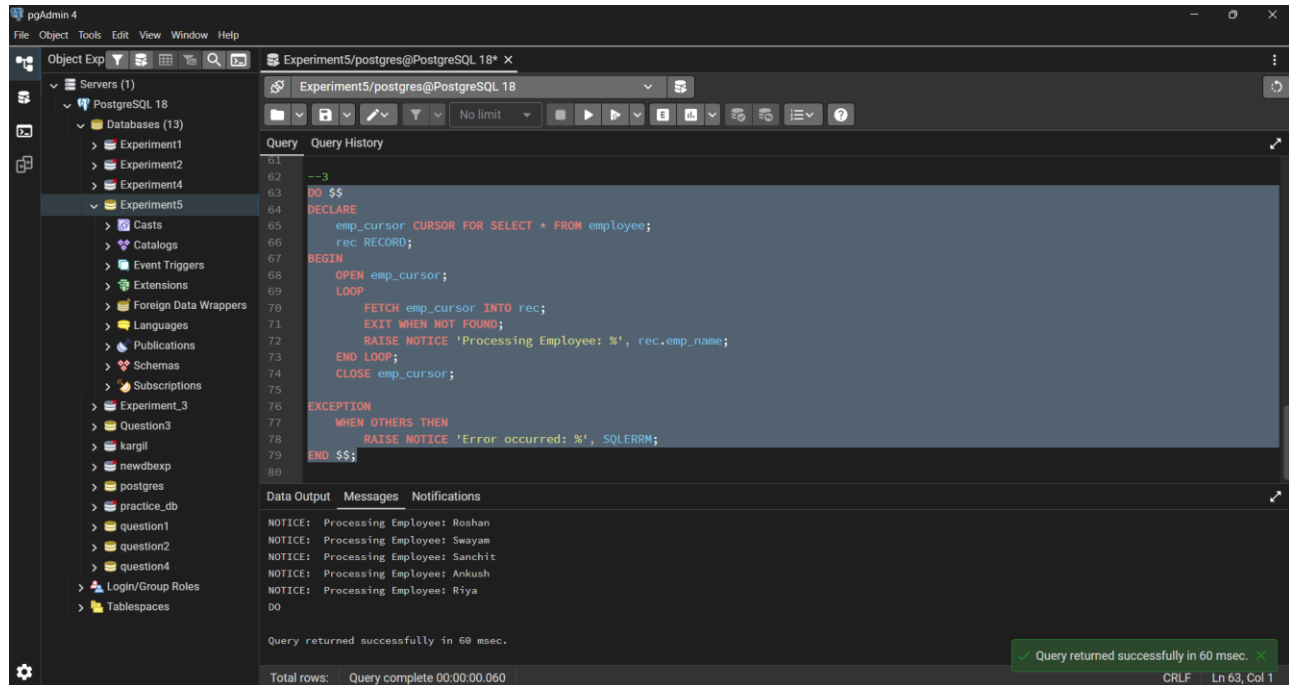
```
46
47
48 IF rec.experience >= 5 AND rec.performance = 'A' THEN
49     new_salary := rec.salary * 1.20;
50 ELSEIF rec.experience >= 3 AND rec.performance = 'B' THEN
51     new_salary := rec.salary * 1.10;
52 ELSE
53     new_salary := rec.salary * 1.05;
54 END IF;
55
56 UPDATE employee
57 SET salary = new_salary
58 WHERE emp_id = rec.emp_id;
59 END LOOP;
60 CLOSE emp_cursor;
61
62 --3
63 DO $$
64 DECLARE
65     emp_cursor CURSOR FOR SELECT * FROM employee;
```

The 'Data Output' tab shows the results of the query, displaying a single row of employee data as a notice:

```
DO
```

The status bar at the bottom indicates: 'Query returned successfully in 59 msec.' and 'Total rows: Query complete 00:00:00.059'.

### Step3: Exception and Status Handling



The screenshot shows the pgAdmin 4 interface. On the left, the 'Object Explorer' shows a tree of databases under 'PostgreSQL 18', with 'Experiment5' selected. The main pane displays a SQL query with the following code:

```
61
62
63 --3
64 DECLARE
65     emp_cursor CURSOR FOR SELECT * FROM employee;
66     rec RECORD;
67 BEGIN
68     OPEN emp_cursor;
69     LOOP
70         FETCH emp_cursor INTO rec;
71         EXIT WHEN NOT FOUND;
72         RAISE NOTICE 'Processing Employee: %', rec.emp_name;
73     END LOOP;
74     CLOSE emp_cursor;
75
76 EXCEPTION
77     WHEN OTHERS THEN
78         RAISE NOTICE 'Error occurred: %', SQLERRM;
79 END $$;
80
```

Below the query editor, the 'Messages' tab shows the following output:

```
NOTICE: Processing Employee: Roshan
NOTICE: Processing Employee: Swayam
NOTICE: Processing Employee: Sanchit
NOTICE: Processing Employee: Ankush
NOTICE: Processing Employee: Riya
DO
```

At the bottom, a green status bar indicates: 'Query returned successfully in 60 msec.' and 'Total rows: Query complete 00:00:00.060'.

### Learning Outcomes:

- Understood cursor-based row-by-row processing
- Learnt cursor lifecycle in PostgreSQL
- Applied complex conditional logic using cursors
- Handled cursor termination and exceptions
- Gained confidence in writing procedural SQL programs.