



Experiment No. 5

Student Name: Roshan Kumar Singh
Branch: MCA(GEN)
Semester: 2nd
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 $$$
DECLARE
    emp_cursor CURSOR FOR
        SELECT emp_id, emp_name, salary FROM employee;
    rec RECORD;
BEGIN
    OPEN emp_cursor;
    LOOP
        FETCH emp_cursor INTO rec;
        EXIT WHEN NOT FOUND;
        RAISE NOTICE 'ID: %, Name: %, Salary: %',
    END LOOP;
END$$;
```



```
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 shows the pgAdmin 4 interface with a query editor window titled "Experiment5/postgres@PostgreSQL 18* X". The left sidebar shows a tree view of databases, including "Experiment1", "Experiment2", "Experiment4", and "Experiment5". The "Query" tab contains the following PostgreSQL code:

```
CREATE TABLE employee (
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR(50),
    salary INT,
    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');
```

The "Data Output" tab shows the result of the INSERT statement:

```
INSERT 0 5
```

Below the query editor, the status bar indicates "Query returned successfully in 71 msec." and "Total rows: 5 Query complete 00:00:00.071".

Step1: Implementing a Simple Forward-Only Cursor

```

pgAdmin 4
File Object Tools Edit View Window Help
Object Exp Servers (1) PostgreSQL 18 Databases (13)
Experiment5 Experiment1 Experiment2 Experiment4
Experiment5 Casts Catalogs Event Triggers Extensions Foreign Data Wrappers Languages Publications Schemas Subscriptions Experiment_3 Question3 kargil newdbexp postgres practice_db question1 question2 question4 Login/Group Roles Tablespaces
Experiment5
Query History
16 --1
17 DO $$*
18 DECLARE
19     emp_cursor CURSOR FOR
20         SELECT emp_id, emp_name, salary FROM employee;
21     rec RECORD;
22 BEGIN
23     OPEN emp_cursor;
24     LOOP
25         FETCH emp_cursor INTO rec;
26         EXIT WHEN NOT FOUND;
27         RAISE NOTICE 'ID: %, Name: %, Salary: %',
28             rec.emp_id, rec.emp_name, rec.salary;
29     END LOOP;
30     CLOSE emp_cursor;
31 END $$;
32
33
Data Output Messages Notifications
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
Query returned successfully in 1 secs 73 msec.

Total rows: 5 Query complete 00:00:01.073 CRLF Ln 32, Col 8

```

Step2: Complex Row-by-Row Manipulation

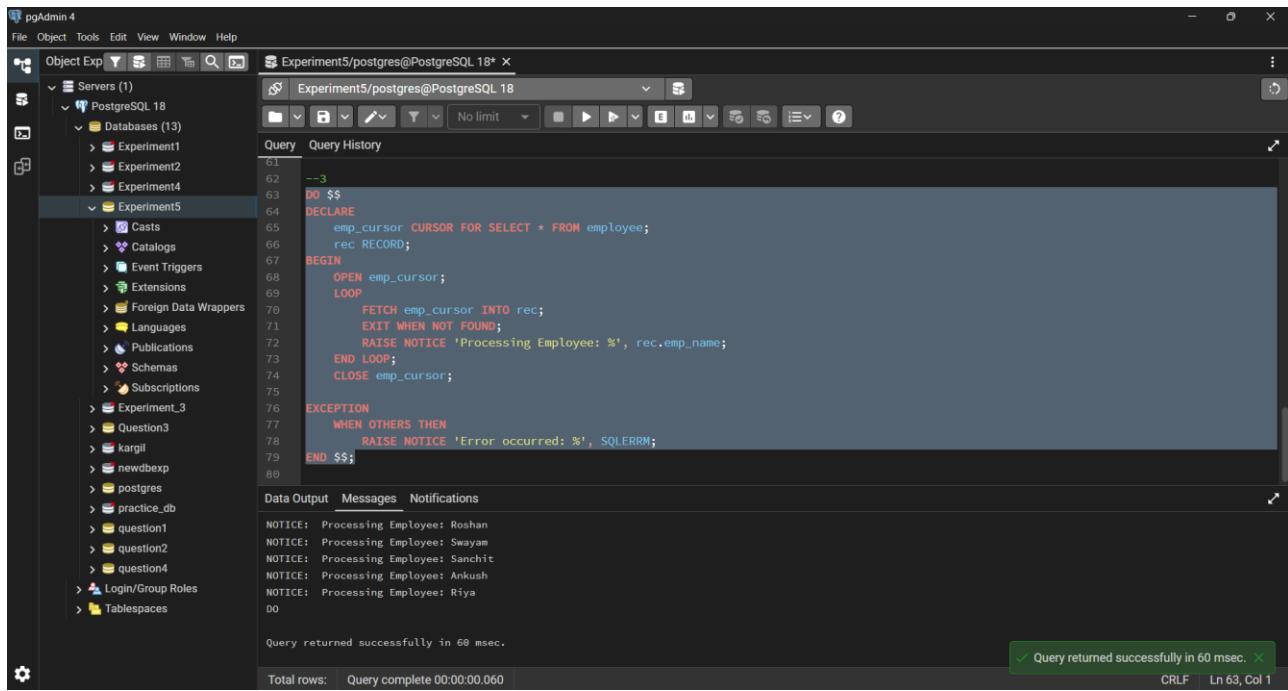
```

pgAdmin 4
File Object Tools Edit View Window Help
Object Exp Servers (1) PostgreSQL 18 Databases (13)
Experiment5 Experiment1 Experiment2 Experiment4
Experiment5 Casts Catalogs Event Triggers Extensions Foreign Data Wrappers Languages Publications Schemas Subscriptions Experiment_3 Question3 kargil newdbexp postgres practice_db question1 question2 question4 Login/Group Roles Tablespaces
Experiment5
Query History
46
47 IF rec.experience >= 5 AND rec.performance = 'A' THEN
48     new_salary := rec.salary * 1.20;
49 ELSIF rec.experience >= 3 AND rec.performance = 'B' THEN
50     new_salary := rec.salary * 1.10;
51 ELSE
52     new_salary := rec.salary * 1.05;
53 END IF;
54
55 UPDATE employee
56 SET salary = new_salary
57 WHERE emp_id = rec.emp_id;
58 END LOOP;
59 CLOSE emp_cursor;
60 END $$;
61
62 --3
63 DO $$*
64 DECLARE
65     emp_cursor CURSOR FOR SELECT * FROM employee;
66
67
Data Output Messages Notifications
DO
Query returned successfully in 59 msec.

Total rows: 5 Query complete 00:00:00.059 CRLF Ln 60, Col 8

```

Step3: Exception and Status Handling



```
--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 $$;

Data Output Messages Notifications
NOTICE: Processing Employee: Roshan
NOTICE: Processing Employee: Swayam
NOTICE: Processing Employee: Sanchit
NOTICE: Processing Employee: Ankush
NOTICE: Processing Employee: Riya
DO

Query returned successfully in 60 msec.

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.