Branch: MCA (Data Science) Kargil	**Semester:** 2
Student Name: Chaitanya Sharma	**UID:** 25MCD10056
Subject Name: Technical Training - I Lab	**Subject Code**: 25CAP-652
Section/Group: 25MCD KAR-1	**Date of Performance:** 27-02-2026

Experiment 5

1. **Aim:** To gain hands-on experience in creating and using cursors for row-by-row processing in a database, enabling sequential access and manipulation of query results for complex business logic. **(Company Tags: Infosys, Wipro, TCS, Capgemini)**

2. **Tools used:** PostgreSQL

3. **Objectives**:

   - **Sequential Data Access:** To understand how to fetch rows one by one from a result set using cursor mechanisms.
   - **Row-Level Manipulation:** To perform specific operations or calculations on individual records that require conditional procedural logic.
   - **Resource Management:** To learn the lifecycle of a cursor: Declaring, Opening, Fetching, and importantly, Closing and Deallocating to manage system memory.
   - **Exception Handling:** To handle cursor-related errors and performance considerations during large-scale data iteration.

4. **Theory:** While SQL is generally set-oriented, certain tasks require a procedural approach where we process one row at a time. This is where **Cursors** are used:

   I. **Cursor Types:** Cursors can be Implicit (managed by the system) or Explicit (defined by the developer). They can also be Forward-Only (moving only toward the end) or Scrollable (moving back and forth).

   II. **The Lifecycle: * DECLARE:** Defines the SQL query for the cursor.
      - **OPEN:** Executes the query and establishes the result set.
      - **FETCH:** Retrieves a specific row into variables for processing.
      - **CLOSE:** Releases the current result set.
      - **DEALLOCATE:** Removes the cursor definition from memory.

   III. **Use Case:** Cursors are ideal for generating row-specific reports, updating balances based on complex historical data, or migrating data where each record needs individual validation.

5. Experiment Steps:

Step 1: Implementing a Simple Forward-Only Cursor

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

Query:

```
-- Query 1
-- Create Staff table
CREATE TABLE IF NOT EXISTS Staff (
    StaffID INT PRIMARY KEY,
    StaffName VARCHAR(50),
    MonthlyPay INT
);
-- Insert sample data
INSERT INTO Staff VALUES (101, 'Dr. Meera', 70000)
ON CONFLICT (StaffID) DO NOTHING;
INSERT INTO Staff VALUES (102, 'Nurse Karan', 45000)
ON CONFLICT (StaffID) DO NOTHING;
INSERT INTO Staff VALUES (103, 'Technician Isha', 38000)
ON CONFLICT (StaffID) DO NOTHING;
-- Cursor block
DO $$
DECLARE
    staff_record RECORD;
    staff_cursor CURSOR FOR
        SELECT StaffID, StaffName, MonthlyPay FROM Staff;
BEGIN
```

```
    OPEN staff_cursor;
    LOOP
        FETCH staff_cursor INTO staff_record;
        EXIT WHEN NOT FOUND;
        RAISE NOTICE 'ID: %, Name: %, Monthly Pay: %',
        staff_record.StaffID,
        staff_record.StaffName,
        staff_record.MonthlyPay;
    END LOOP;
    CLOSE staff_cursor;
END $$;
```

Output:

```
NOTICE:  ID: 101, Name: Dr. Meera, Monthly Pay: 70000
NOTICE:  ID: 102, Name: Nurse Karan, Monthly Pay: 45000
NOTICE:  ID: 103, Name: Technician Isha, Monthly Pay: 38000
DO
```

Step 2: Complex Row-by-Row Manipulation

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

Query:

```
-- Query 2
-- Add YearsOfService column
ALTER TABLE Staff
ADD COLUMN IF NOT EXISTS YearsOfService INT;
```

```
-- Update sample service values
UPDATE Staff SET YearsOfService = 3 WHERE StaffID = 101;
UPDATE Staff SET YearsOfService = 6 WHERE StaffID = 102;
UPDATE Staff SET YearsOfService = 9 WHERE StaffID = 103;
-- Cursor to update salary based on years of service
DO $$
DECLARE
    staff_record RECORD;
    staff_cursor CURSOR FOR
        SELECT StaffID, MonthlyPay, YearsOfService FROM Staff;
BEGIN
    OPEN staff_cursor;
    LOOP
        FETCH staff_cursor INTO staff_record;
        EXIT WHEN NOT FOUND;
        -- Salary update logic
        IF staff_record.YearsOfService >= 8 THEN
            UPDATE Staff
            SET MonthlyPay = MonthlyPay + 8000
            WHERE StaffID = staff_record.StaffID;
        ELSIF staff_record.YearsOfService >= 5 THEN
            UPDATE Staff
            SET MonthlyPay = MonthlyPay + 5000
```

```
        WHERE StaffID = staff_record.StaffID;
    ELSE
        UPDATE Staff
        SET MonthlyPay = MonthlyPay + 2000
        WHERE StaffID = staff_record.StaffID;
    END IF;
  END LOOP;
  CLOSE staff_cursor;
END $$;
-- View updated table
SELECT * FROM Staff;
```

Output:

	staffid [PK] integer	staffname character varying (50)	monthlypay integer	yearsofservice integer
1	101	Dr. Meera	72000	3
2	102	Nurse Karan	50000	6
3	103	Technician Isha	46000	9

Step 3: Exception and Status Handling

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

Query:

```
-- Query 3
DO $$
```

```
DECLARE
    staff_record RECORD;
    staff_cursor CURSOR FOR
        SELECT StaffID, StaffName, MonthlyPay FROM Staff;
BEGIN
    OPEN staff_cursor;
    -- Check if cursor has data
    FETCH staff_cursor INTO staff_record;
    IF NOT FOUND THEN
        RAISE NOTICE 'No records found in Staff table.';
    ELSE
        LOOP
            RAISE NOTICE 'Processing Staff ID: %, Name: %, Monthly Pay: %',
            staff_record.StaffID,
            staff_record.StaffName,
            staff_record.MonthlyPay;
            FETCH staff_cursor INTO staff_record;
            EXIT WHEN NOT FOUND;
        END LOOP;
    END IF;
    CLOSE staff_cursor;
EXCEPTION
    WHEN OTHERS THEN
```

RAISE NOTICE 'An error occurred: %', SQLERRM;

END $$;

Output:

```
NOTICE:  Processing Staff ID: 101, Name: Dr. Meera, Monthly Pay: 72000
NOTICE:  Processing Staff ID: 102, Name: Nurse Karan, Monthly Pay: 50000
NOTICE:  Processing Staff ID: 103, Name: Technician Isha, Monthly Pay: 46000
DO
```

6. Learning Outcomes:

- Understood how to declare, open, fetch, loop through, and close a cursor in PL/pgSQL
- Applied conditional logic inside a cursor to update table records dynamically.
- Implemented basic exception handling to manage runtime errors in procedural SQL blocks.