Branch: MCA (Data Science)	**Semester:** 2
Student Name: Adarsh Mishra	**UID:** 25MCD10065
Subject Name: Technical Training - I Lab	**Subject Code**: 25CAP-652
Section/Group: 25MCD KAR-1	**Date of Performance:** 24 Februrary,2026

EXPERIMENT – 05

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)**

Tools Used

- PostgreSQL

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.

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:

1. 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).
2. 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.
3. 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

---

.

Experiment Steps:

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.

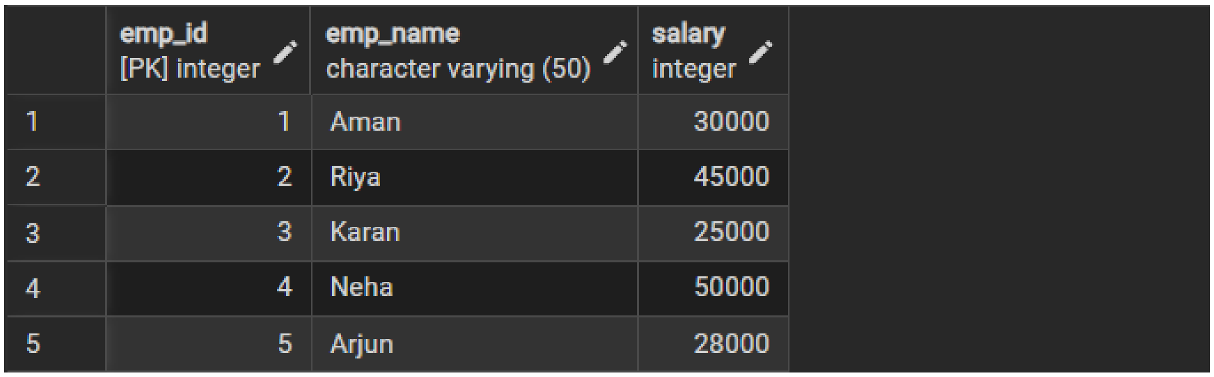
Query:

Creating Table:

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

-- Step 2: Insert Sample Data
INSERT INTO employee VALUES
(1,'Aman',30000),
(2,'Riya',45000),
(3,'Karan',25000),
(4,'Neha',50000),
(5,'Arjun',28000);
```

Output:



	emp_id [PK] integer	emp_name character varying (50)	salary integer
1	1	Aman	30000
2	2	Riya	45000
3	3	Karan	25000
4	4	Neha	50000
5	5	Arjun	28000

Cursor Procedure:

```
CREATE OR REPLACE PROCEDURE update_salary_cursor()
LANGUAGE plpgsql
AS $$
DECLARE
emp_record RECORD;
emp_cursor CURSOR FOR
SELECT emp_id, salary FROM employee;
BEGIN

OPEN emp_cursor;
```

```
LOOP
FETCH emp_cursor INTO emp_record;
EXIT WHEN NOT FOUND;

-- Business Logic
IF emp_record.salary < 40000 THEN
UPDATE employee
SET salary = salary + (salary * 0.10)
WHERE emp_id = emp_record.emp_id;
END IF;

END LOOP;

CLOSE emp_cursor;

END;
$$;
```

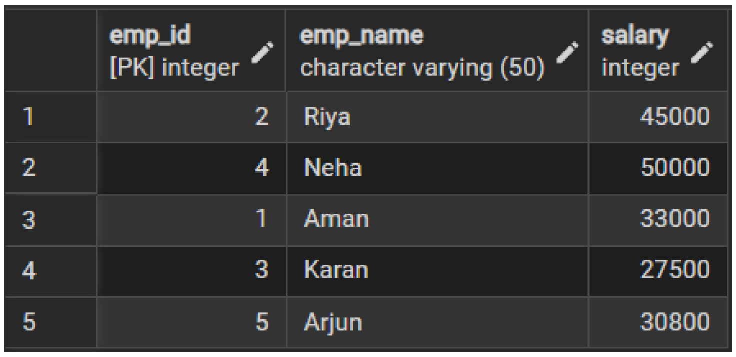
Call Procedure:

```
CALL update_salary_cursor();
```

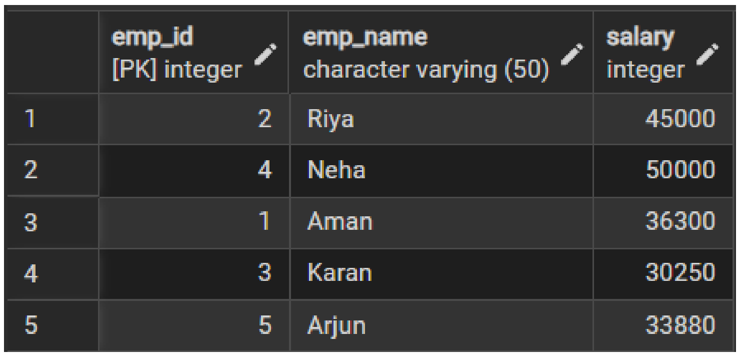
Displaying Result:

```
SELECT * FROM employee;
```

Output:



	emp_id [PK] integer	emp_name character varying (50)	salary integer
1	2	Riya	45000
2	4	Neha	50000
3	1	Aman	33000
4	3	Karan	27500
5	5	Arjun	30800



	emp_id [PK] integer	emp_name character varying (50)	salary integer
1	2	Riya	45000
2	4	Neha	50000
3	1	Aman	36300
4	3	Karan	30250
5	5	Arjun	33880

Learning Outcomes:

- Understand the concept and purpose of **cursors in PostgreSQL**.
- Create and use **PL/pgSQL procedures** with cursors.
- Perform **row-by-row processing** on query results.
- Apply **conditional logic and data manipulation** while fetching records from a cursor.
- Gain practical experience in implementing **business logic inside a database**.