Worksheet No. 3

Student Name: ANINDITA DHAR **UID:** 25MCA20259
Branch: MCA (GENERAL) **Section/Group:** MCA- 1-A
Semester: II **Date of Performance:** 27/01/2026

Subject Name: Technical Training **Subject Code:** 25CAP-652

1.Aim/Overview of the practical:

To implement conditional decision-making logic in PostgreSQL using **IF–ELSE constructs** and **CASE expressions** for classification, validation, and rule-based data processing.

2. Objectives:

- To understand conditional execution in SQL.
- To implement decision-making logic using CASE expressions.
- To simulate real-world rule validation scenarios.
- To classify data based on multiple conditions.
- To strengthen SQL logic skills required in interviews and backend systems.

3. Input/Apparatus Used:

- PostgreSQL
- pgAdmin

4. Practical / Experiment Steps

- Create a table that stores:
  - A unique identifier
  - A schema or entity name
  - A numeric count representing violations or issues
  - Populate the table with multiple records having different violation counts.

- Classifying data using a CASE expression

  - Retrieve schema names and their violation counts.
  - Use conditional logic to classify each schema into categories such as:
    - No Violation
    - Minor Violation

- Moderate Violation
- Critical Violation

➢ Applying CASE Logic in Data Updates

- Add a new column to store approval status.
- Update this column based on violation count using conditional rules such as:
  a) Approved
  b) Needs Review
  c) Rejected

➢ Implementing if-else logic using PL/SQL

- Use a procedural block instead of a SELECT statement.
- Declare a variable representing violation count.
- Display different messages based on the value of the variable using IF–ELSE logic.

➢ Create a table to store student names and marks.
➢ Classify students into grades based on their marks using conditional logic.

➢ Using CASE for custom sorting

- Retrieve schema details.
- Apply conditional priority while sorting records based on violation severity.

4.Procedure/Algorithm/Code:

i.Start the system and log in to the computer.
ii. Open PostgreSQL software.

iii. **Create a table using the DDL command.**

```
CREATE TABLE Schema_Analysis (
schema_id SERIAL PRIMARY KEY,
schema_name VARCHAR(100),
violation_count INT
);
```

iv. Insert records into the table named customer_orders.

```
INSERT INTO Schema_Analysis VALUES(1,'Employee_Schema',0);
INSERT INTO Schema_Analysis VALUES(2,'Payroll_Schema',2);
INSERT INTO Schema_Analysis VALUES(3,'Inventory_Schema',5);
INSERT INTO Schema_Analysis VALUES(4,'Finance_Schema',1);
```

vi. Display all records.

```
SELECT * FROM Schema_Analysis;
```

	schema_id [PK] integer	schema_name character varying (100)	violation_count integer
1	1	Employee_Schema	0
2	2	Payroll_Schema	2
3	3	Inventory_Schema	5
4	4	Finance_Schema	1

vii. Classifying data using a CASE expression

```
SELECT
   schema_id,
   schema_name,
   violation_count,
   CASE
      WHEN violation_count = 0 THEN 'No Violation'
      WHEN violation_count BETWEEN 1 AND 2 THEN 'Minor Violation'
      ELSE 'Critical Violation'
   END AS Violation_category
FROM Schema_Analysis;
```

	schema_id [PK] integer	schema_name character varying (100)	violation_count integer	violation_category text
1	1	Employee_Schema	0	No Violation
2	2	Payroll_Schema	2	Minor Violation
3	3	Inventory_Schema	5	Critical Violation
4	4	Finance_Schema	1	Minor Violation

viii. Applying CASE Logic in Data Updates

```
ALTER TABLE Schema_Analysis
ADD Status VARCHAR(100);

UPDATE Schema_Analysis
SET Status = CASE
        WHEN violation_count = 0 THEN 'APPROVED'
        WHEN violation_count BETWEEN 1 AND 2 THEN 'REVIEW'
        ELSE 'REJECTED'
      END;
```

	schema_id [PK] integer	schema_name character varying (100)	violation_count integer	status character varying (100)
1	1	Employee_Schema	0	APPROVED
2	2	Payroll_Schema	2	REVIEW
3	3	Inventory_Schema	5	REJECTED
4	4	Finance_Schema	1	REVIEW

ix. Implementing if-else logic using PL/SQL

```
DO $$
DECLARE
   VAL INT := 2;
BEGIN
   IF VAL > 10 THEN
      RAISE NOTICE 'Value greater than 10';
   ELSIF VAL BETWEEN 10 AND 50 THEN
      RAISE NOTICE 'Value is between 10 and 50';
   ELSE
      RAISE NOTICE 'Value less than 10';
   END IF;
END $$;
```

```
60
Data Output  Messages  Notifications

NOTICE:  Value less than 10
DO

Query returned successfully in 80 msec.
```

x. Create a table to store student names and marks.

```
CREATE TABLE Students_1(

student_names VARCHAR (100),

student_marks INT

);
```

xi. Classify students into grades based on their marks using conditional logic.

```
CREATE TABLE IF NOT EXISTS students_12(
   id SERIAL PRIMARY KEY,
   name VARCHAR(100),
   marks INTEGER CHECK (marks >= 0 AND marks <= 100),
   grade VARCHAR(2)
);

INSERT INTO students_12(name, marks) VALUES ('Anindita', 85);
INSERT INTO students_12(name, marks) VALUES('John', 92);
INSERT INTO students_12(name, marks) VALUES ('Alice', 65);
INSERT INTO students_12(name, marks)VALUES('Bob', 45);
INSERT INTO students_12(name, marks) VALUES('Eve', 78);

DO $$
DECLARE
   student_rec RECORD;
   student_grade VARCHAR(2);
BEGIN
   -- Loop through students and assign grades
   FOR student_rec IN SELECT id, name, marks FROM students_12 LOOP

      -- Conditional logic for grades
      CASE
         WHEN student_rec.marks >= 90 THEN student_grade := 'A+';
         WHEN student_rec.marks >= 80 THEN student_grade := 'A';
         WHEN student_rec.marks >= 70 THEN student_grade := 'B';
         WHEN student_rec.marks >= 60 THEN student_grade := 'C';
         WHEN student_rec.marks >= 50 THEN student_grade := 'D';
         ELSE student_grade := 'F';
      END CASE;


      -- Update grade
      UPDATE students_12
```

```
        SET grade = student_grade
        WHERE id = student_rec.id;

        RAISE NOTICE 'Student: %, Marks: %, Grade: %',
                student_rec.name, student_rec.marks, student_grade;
    END LOOP;
END $$;
```

```
NOTICE:  Student: Anindita, Marks: 85, Grade: A
NOTICE:  Student: John, Marks: 92, Grade: A+
NOTICE:  Student: Alice, Marks: 65, Grade: C
NOTICE:  Student: Bob, Marks: 45, Grade: F
NOTICE:  Student: Eve, Marks: 78, Grade: B
DO

Query returned successfully in 116 msec.
```

5. I/O Analysis (Input / Output Analysis)

- Creates 3 tables (Schema_Analysis, students_12 populated).
- Processes schema violations → status.
- Outputs 7 RAISE NOTICE messages.
- No errors; idempotent elements like IF NOT EXISTS and ON CONFLICT DO NOTHING handle re-runs safely.

6. Learning Outcomes:

- CASE Expression (SQL Level)
- UPDATE with CASE
- Loops & Record Processing
- Data Integrity