

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

 vViolation INT := 0;

BEGIN

 IF vViolation = 0 THEN

 RAISE NOTICE 'Schema is clean and vViolation value is %', vViolation;

 ELSIF vViolation <= 2 THEN

 RAISE NOTICE 'Schema has minor issues';

 ELSE

 RAISE NOTICE 'Schema has critical violations';

 END IF;

END \$\$;

```
NOTICE: Schema is clean and vViolation value is 0
DO
```

```
Query returned successfully in 149 msec.
```

x. Custom sorting based on priority.

```
SELECT schema_name, violation_count
FROM Schema_Analysis
ORDER BY
CASE
    WHEN violation_count = 0 THEN 1
    WHEN violation_count <= 2 THEN 2
    ELSE 3
END;
```

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

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

- Creates table (Schema_Analysis).
- Processes schema violations → status.
- Outputs show 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