Experiment 3

Student Name: Sumanyu Seth **UID: 25MCI10040**
Branch: MCA (AI & ML) **Section/Group: 25MAM_KAR-1_A**
Semester: II **Date of Performance: 27/01/26**
Subject Name: Technical Training **Subject Code: 25CAP-652**

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.

Software Requirements:

- PostgreSQL

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 system

Theory

In real-world database systems, data often needs to be validated, categorized, or transformed based on business rules. Conditional logic allows the database to make decisions dynamically instead of relying solely on application-layer logic.

PostgreSQL supports conditional logic mainly through:

- CASE Expressions (used inside SELECT, UPDATE, INSERT)
- IF–ELSE constructs (used inside PL/pgSQL blocks such as functions and procedures)

CASE Expression

- Evaluates conditions sequentially
- Returns a value based on the first true condition
- Can be used in SELECT, UPDATE, ORDER BY, and WHERE clauses

Types of CASE

- Simple CASE → compares expressions
- Searched CASE → evaluates boolean conditions

Conditional logic is heavily used in:

- Data classification (grades, salary slabs)
- Violation detection
- Status mapping
- Business rule enforcement

Companies like Amazon, SAP, Oracle, and Adobe frequently test CASE-based logic in SQL interviews.

Experiment / Practical Steps:

Prerequisite Understanding
Students should first 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.

Step 1: Classifying Data Using CASE Expression
Task for Students:

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

Learning Focus:

- Using **searched CASE**
- Sequential condition checking
- Real-world compliance reporting logic

Step 2: Applying CASE Logic in Data Updates
Task for Students:

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

- Rejected

Learning Focus:

- Automating decisions inside the database
- Reducing application-side logic
- Using CASE inside UPDATE statements

Step 3: Implementing IF–ELSE Logic Using PL/pgSQL
Task for Students:

- 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.

Learning Focus:

- Understanding procedural SQL
- ELSE-IF ladder execution
- Backend validation logic in stored procedures

Step 4: Real-World Classification Scenario (Grading System)
Task for Students:

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

Learning Focus:

- Common interview use case
- Data categorization
- Rule-based evaluation

Step 5: Using CASE for Custom Sorting
Task for Students:

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

Learning Focus:

- Advanced CASE usage

Custom ordering logic and Dashboard and reporting scenarios

Practical / Experiment Steps:

```
create table my_schemas (
   id serial primary key,
   schema_name varchar(50),
   violation_count int
);
```

```
insert into my_schemas (schema_name, violation_count) values
('alpha', 0),
('beta', 2),
('gamma', 5),
('delta', 10),
 ('epsilon', 1);

SELECT * from my_schemas;
```

id [PK] integer	schema_name character varying (50)	violation_count integer
1	alpha	0
2	beta	2
3	gamma	5
4	delta	10
5	epsilon	1

Step 1 : Classifying data using case expression

```
select
    schema_name,
    violation_count,
    case
        when violation_count = 0 then 'no violation'
        when violation_count between 1 and 3 then 'minor violation'
        else 'critical violation'
    end as violation_status
from my_schemas;
```

schema_name character varying (50)	violation_count integer	violation_status text
alpha	0	no violation
beta	2	minor violation
gamma	5	critical violation
delta	10	critical violation
epsilon	1	minor violation

Step 2: Applying CASE Logic in Data Updates

```
update my_schemas
set approval_status = case
    when violation_count = 0 then 'approved'
    when violation_count between 1 and 3 then 'needs review'
    else 'rejected'
end;
select * from my_schemas;
```

id [PK] integer	schema_name character varying (50)	violation_count integer	approval_status character varying (20)
1	alpha	0	approved
2	beta	2	needs review
3	gamma	5	rejected
4	delta	10	rejected
5	epsilon	1	needs review

Step 3: Implementing IF–ELSE Logic Using PL/pgSQL

```
do $$
declare
   v_count int := 3;
begin
   if v_count = 0 then
      raise notice 'no violation';
   elsif v_count between 1 and 3 then
      raise notice 'minor violation';
   else
      raise notice 'critical violation';
   end if;
end;
$$;
```

```
NOTICE:  minor violation
DO
```

```
do $$
declare
   v_count int := 0;
begin
   if v_count = 0 then
      raise notice 'no violation';
   elsif v_count between 1 and 3 then
      raise notice 'minor violation';
   else
      raise notice 'critical violation';
   end if;
end;
$$;
```

```
NOTICE:  no violation
DO
```

```
do $$
declare
   v_count int := 6;
begin
   if v_count = 0 then
      raise notice 'no violation';
   elsif v_count between 1 and 3 then
      raise notice 'minor violation';
   else
      raise notice 'critical violation';
   end if;
end;
$$;
```

```
NOTICE:  critical violation
DO
```

Step 4: Grading system

```
create table student_marks (
   id serial primary key,
   student_name varchar(50),
   marks int
);
```

```
insert into student_marks (student_name, marks) values
('amit', 85),
('bobby', 60),
('chaman', 40),
('daksh', 95);
select
    student_name,
    marks,
    case
        when marks >= 80 then 'grade a'
        when marks >= 60 then 'grade b'
        when marks >= 40 then 'grade c'
        else 'fail'
    end as grade
from student_marks;
```

student_name character varying (50)	marks integer	grade text
amit	85	grade a
bobby	60	grade b
chaman	40	grade c
daksh	95	grade a

Step 5: Custom sorting

```
select
    schema_name,
    violation_count
from my_schemas
order by case
    when violation_count = 0 then 1
    when violation_count between 1 and 3 then 2
    else 3
end;
```

schema_name character varying (50)	violation_count integer
alpha	0
beta	2
epsilon	1
gamma	5
delta	10

Learning Outcomes:

- Understand how to classify records using simple case expressions.
- Learn to automate approval logic directly inside update statements.
- Practice writing if–else blocks in pgsql for backend validation.
- Apply conditional rules to grade students based on marks.
- Explore custom sorting with case for reporting and dashboards.