Branch: MCA (Data Science) Kargil	**Semester:** 2
Student Name: Sameer Shahi	**UID:** 25MCD10025
Subject Name: Technical Training - I Lab	**Subject Code**: 25CAP-652
Section/Group: A	**Date of Performance:** 27-01-2026

EXPERIMENT – 03

Implementation of Conditional Logic using IF–ELSE and CASE Statements in PostgreSQL

Aim

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

Tools Used

- 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 systems

---

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
    - Moderate 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:

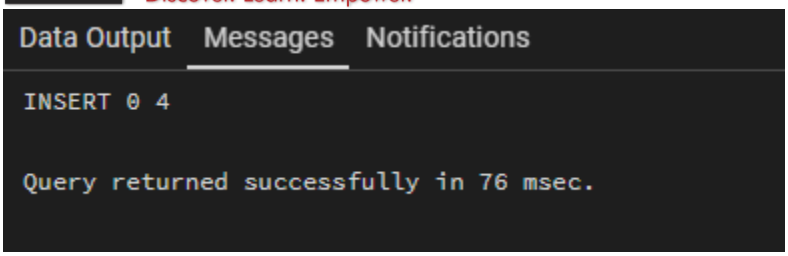
- Advanced CASE usage
- Custom ordering logic
- Dashboard and reporting scenarios

Queries and Output

Prerequisite Queries:

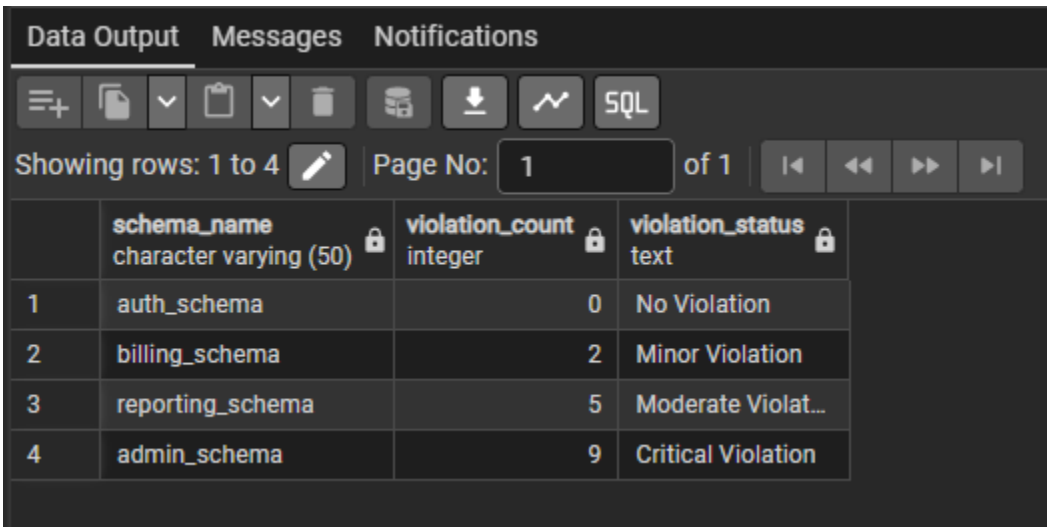
```
create table schema_violations (
    schema_id serial primary key,
    schema_name varchar(50),
    violation_count int
);


insert into schema_violations (schema_name, violation_count) values
('auth_schema', 0),
('billing_schema', 2),
('reporting_schema', 5),
('admin_schema', 9);
```



Step 1 Queries:

```
select
    schema_name,
    violation_count,
    case
        when violation_count = 0 then 'No Violation'
        when violation_count between 1 and 3 then 'Minor Violation'
        when violation_count between 4 and 6 then 'Moderate Violation'
        else 'Critical Violation'
    end as violation_status
from schema_violations;
```



	schema_name character varying (50)	violation_count integer	violation_status text
1	auth_schema	0	No Violation
2	billing_schema	2	Minor Violation
3	reporting_schema	5	Moderate Violat...
4	admin_schema	9	Critical Violation

Step 2 Queries:

```
-- Add a new column to store approval status
alter table schema_violations
```

add column approval_status varchar(20);

Data Output | Messages | Notifications

```
ALTER TABLE

Query returned successfully in 477 msec.
```

-- Update the new column based on violation count using conditional rules

```
update schema_violations
set approval_status =
    case
        when violation_count = 0 then 'Approved'
        when violation_count between 1 and 3 then 'Needs Review'
        else 'Rejected'
    end;
```

Data Output | Messages | Notifications

```
UPDATE 4

Query returned successfully in 241 msec.
```

-- check new column

```
select * from schema_violations;
```

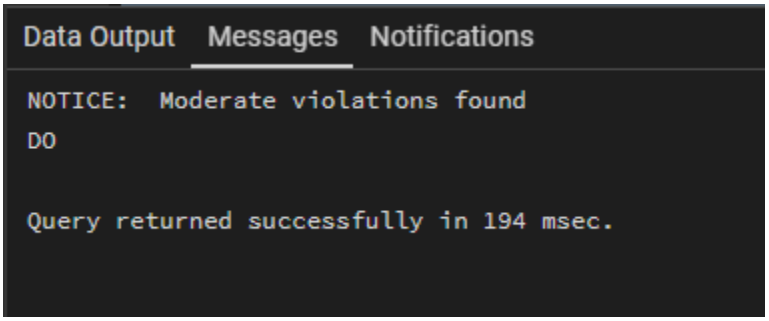
Data Output | Messages | Notifications

Showing rows: 1 to 4 | Page No: 1 of 1

	schema_id [PK] integer	schema_name character varying (50)	violation_count integer	approval_status character varying (20)
1	1	auth_schema	0	Approved
2	2	billing_schema	2	Needs Review
3	3	reporting_schema	5	Rejected
4	4	admin_schema	9	Rejected

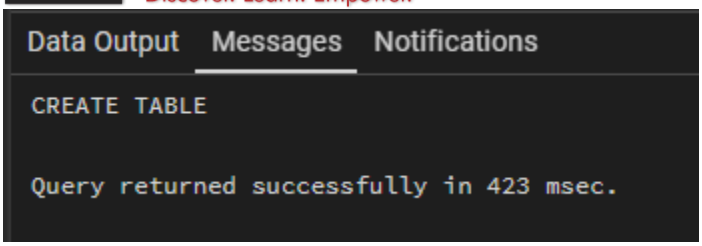
Step 3 Queries:

```
do $$
declare
   v_count int := 5;
begin
   if v_count = 0 then
      raise notice 'No violations detected';
   elsif v_count <= 3 then
      raise notice 'Minor violations found';
   elsif v_count <= 6 then
      raise notice 'Moderate violations found';
   else
      raise notice 'Critical violations found';
   end if;
end $$;
```

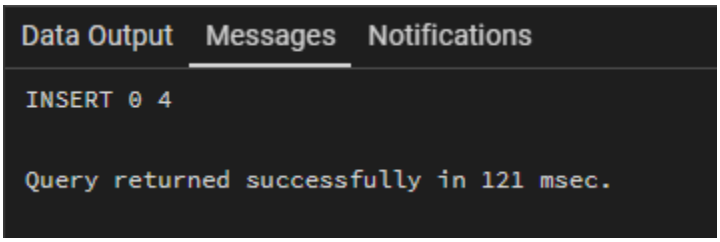


Step 4 Queries:

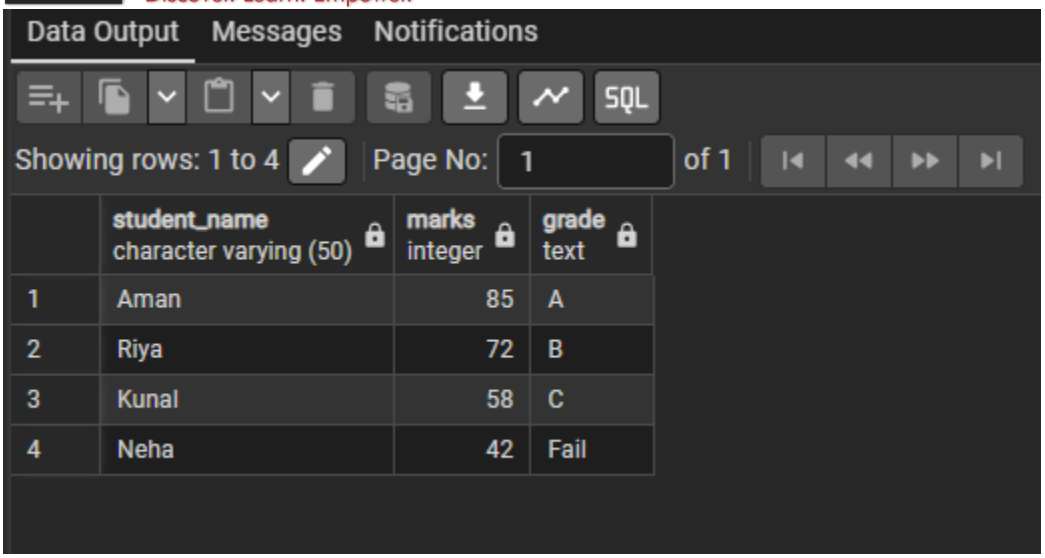
```
-- Create a table to store student names and marks
create table students (
   student_id serial primary key,
   student_name varchar(50),
   marks int
);
```




```
insert into students (student_name, marks) values
('Aman', 85),
('Riya', 72),
('Kunal', 58),
('Neha', 42);
```



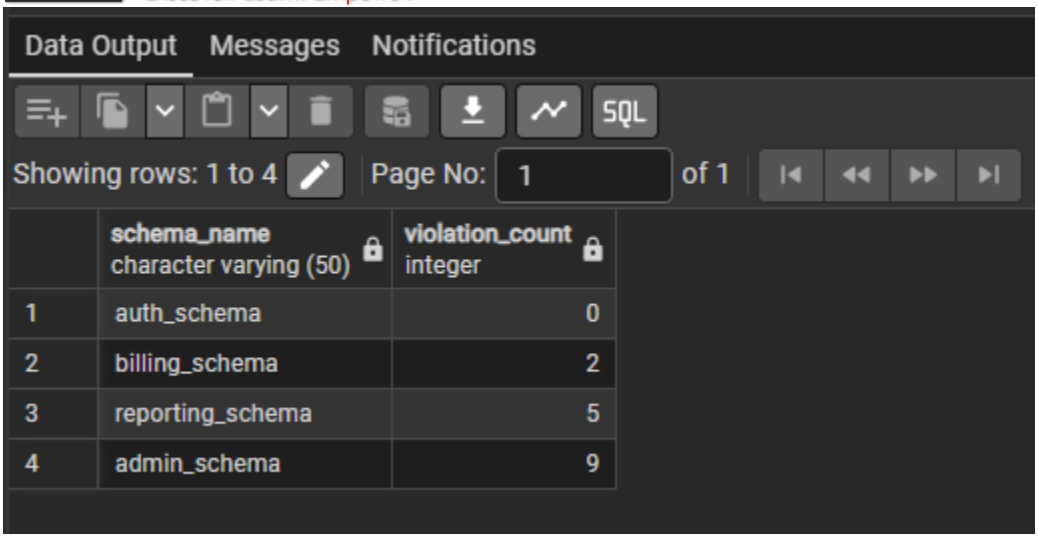

```
-- Classify students into grades based on their marks using conditional logic
select
    student_name,
    marks,
    case
        when marks >= 80 then 'A'
        when marks >= 60 then 'B'
        when marks >= 50 then 'C'
        else 'Fail'
    end as grade
from students;
```



	student_name character varying (50)	marks integer	grade text
1	Aman	85	A
2	Riya	72	B
3	Kunal	58	C
4	Neha	42	Fail

Step 5 Queries:

```
-- Using CASE for Custom Sorting
select
    schema_name,
    violation_count
from schema_violations
order by
    case
        when violation_count = 0 then 1
        when violation_count between 1 and 3 then 2
        when violation_count between 4 and 6 then 3
        else 4
    end;
```



Data Output | Messages | Notifications

Showing rows: 1 to 4 | Page No: 1 of 1

	schema_name character varying (50)	violation_count integer
1	auth_schema	0
2	billing_schema	2
3	reporting_schema	5
4	admin_schema	9

Learning Outcome

This experiment demonstrates how conditional logic is implemented in PostgreSQL using **CASE expressions** and **IF–ELSE constructs**.
Students gain strong command over **rule-based SQL logic**, which is essential for:

- Backend systems
- Analytics
- Compliance reporting
- Placement and technical interviews