Worksheet 3

Student Name:Aryan Tomar **UID:25MCI10167**
Branch:MCA (AI&ML) **Section/Group:1/A**
Semester:2nd **Date of Performance:27/01/2026**
Subject Name:- DBMS LAB **Subject Code:**

1. Aim of the Session

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

2. Software Requirements

- PostgreSQL (Database Server)
- pgAdmin
- Windows Operating System

3. Objective of the Session

After completing this practical, the student will be able to:

- 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

4. Practical / Experiment 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.

5. Procedure of the Practical

(i) Start the system and log in to the computer.

(ii) Open PostgreSQL software.

iii) Create and select the database.

```
Create database Practical3;
```

(iv) Create table using DDL command.

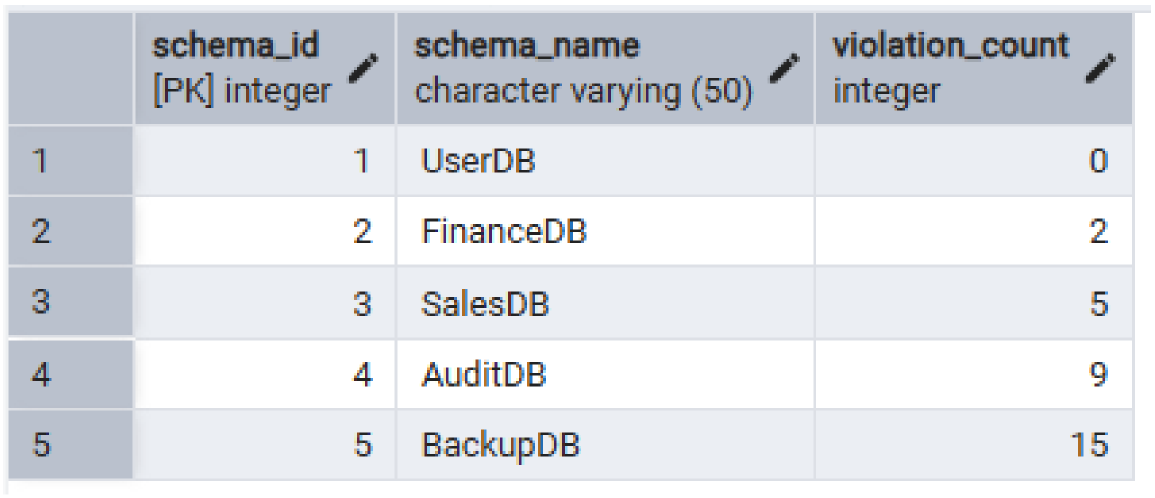
```
create table schema_audit(

schema_id serial primary key,

schema_name varchar(50),

violation_count int

);
```

(v) Insert records into the table.

```
insert into schema_audit(schema_name, violation_count) values
('UserDB', 0),
('FinanceDB', 2),
('SalesDB', 5),
('AuditDB', 9),
('BackupDB', 15);
```

(vi) Display all records.

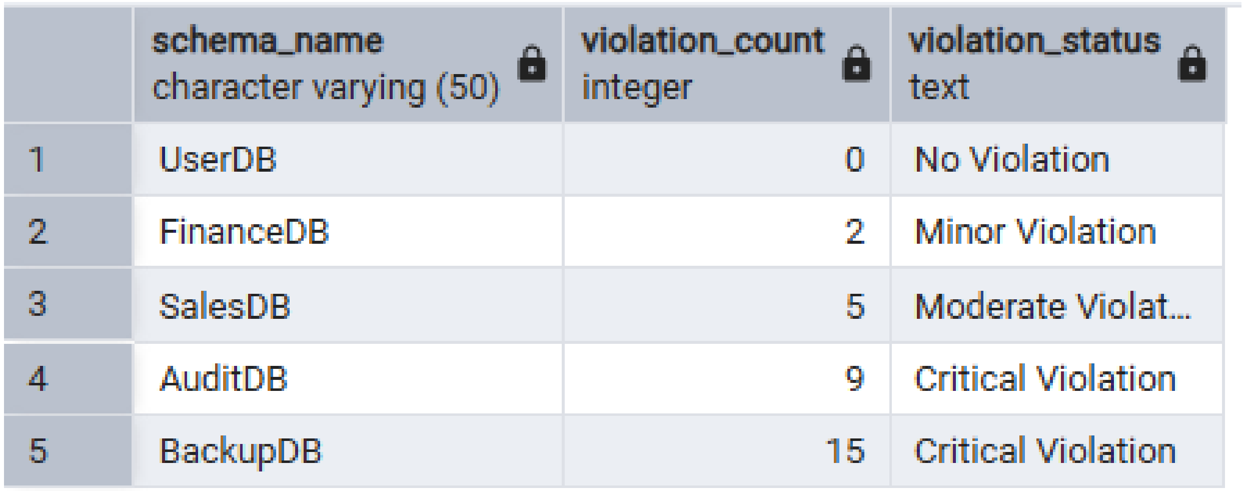
```
select*from schema_audit;
```



	schema_id [PK] integer	schema_name character varying (50)	violation_count integer
1	1	UserDB	0
2	2	FinanceDB	2
3	3	SalesDB	5
4	4	AuditDB	9
5	5	BackupDB	15

Step 1: Classifying Data Using CASE Expression

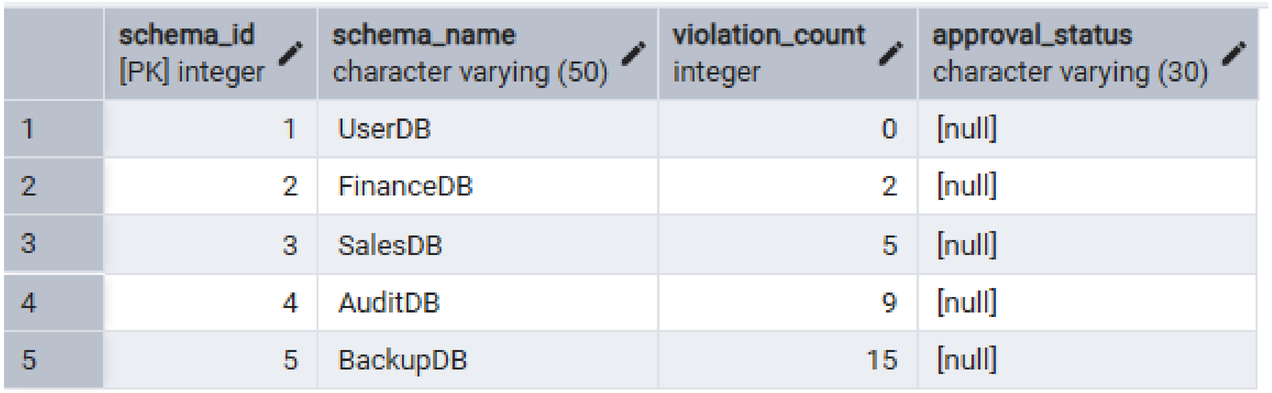
```
select schema_name, violation_count,
case
 when violation_count = 0 then 'No Violation'
 when violation_count between 1 and 3 then 'Minor Violation'
 when violation_count between 4 and 7 then 'Moderate Violation'
 else 'Critical Violation'
end as violation_status
from schema_audit;
```



	schema_name character varying (50)	violation_count integer	violation_status text
1	UserDB	0	No Violation
2	FinanceDB	2	Minor Violation
3	SalesDB	5	Moderate Violat...
4	AuditDB	9	Critical Violation
5	BackupDB	15	Critical Violation

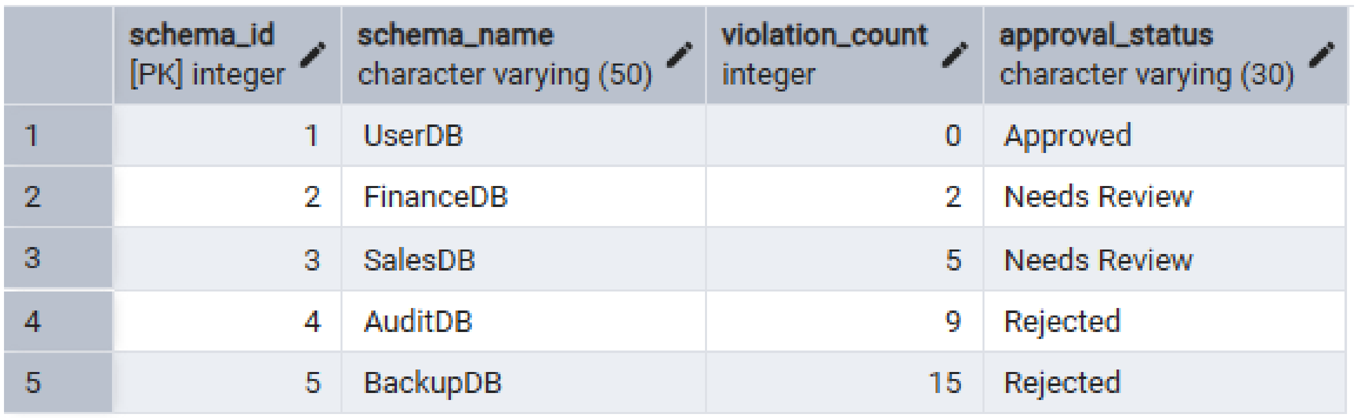
Step 2: Applying CASE Logic in Data Updates

```
alter table schema_audit add approval_status varchar(30);
```



	schema_id [PK] integer	schema_name character varying (50)	violation_count integer	approval_status character varying (30)
1	1	UserDB	0	[null]
2	2	FinanceDB	2	[null]
3	3	SalesDB	5	[null]
4	4	AuditDB	9	[null]
5	5	BackupDB	15	[null]

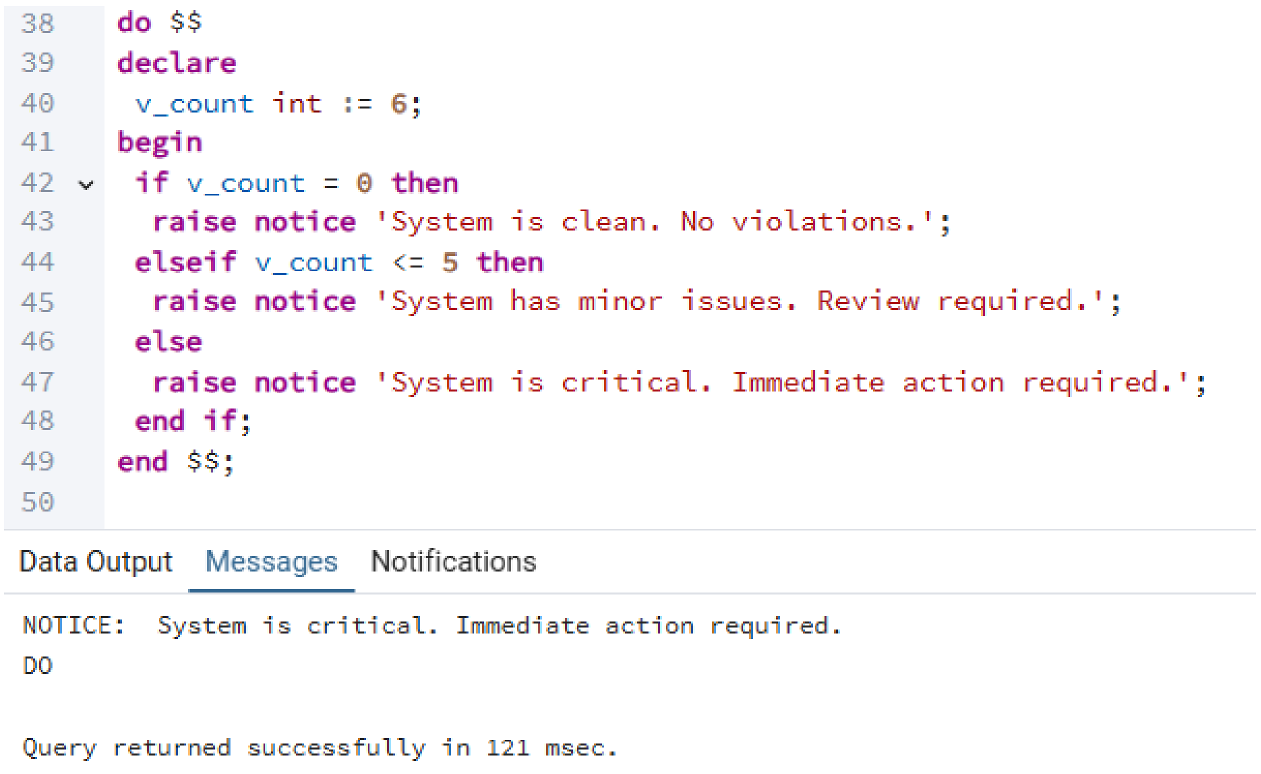
```
update schema_audit

set approval_status =

case

 when violation_count = 0 then 'Approved'

 when violation_count between 1 and 5 then 'Needs Review'

 else 'Rejected'

end;
```



	schema_id [PK] integer	schema_name character varying (50)	violation_count integer	approval_status character varying (30)
1	1	UserDB	0	Approved
2	2	FinanceDB	2	Needs Review
3	3	SalesDB	5	Needs Review
4	4	AuditDB	9	Rejected
5	5	BackupDB	15	Rejected

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

```
do $$
declare
 v_count int := 6;
begin
 if v_count = 0 then
  raise notice 'System is clean. No violations.';
 elseif v_count <= 5 then
  raise notice 'System has minor issues. Review required.';
 else
  raise notice 'System is critical. Immediate action required.';
 end if;
end $$;
```



```
do $$
declare
 v_count int := 6;
begin
 if v_count = 0 then
  raise notice 'System is clean. No violations.';
 elseif v_count <= 5 then
  raise notice 'System has minor issues. Review required.';
 else
  raise notice 'System is critical. Immediate action required.';
 end if;
end $$;
```

```
NOTICE:  System is critical. Immediate action required.
DO

Query returned successfully in 121 msec.
```

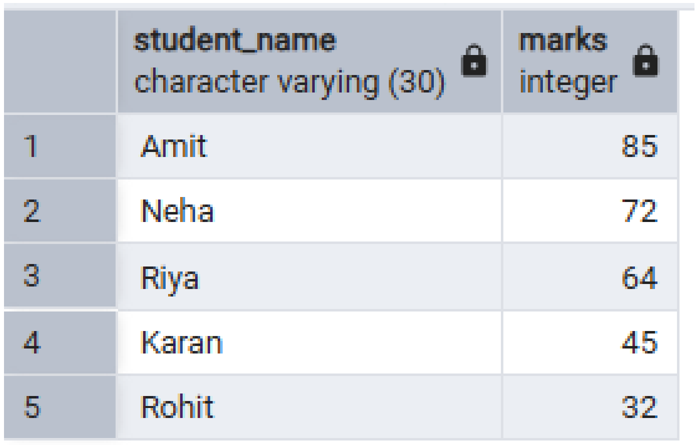
Step 4: Real-World Classification Scenario (Grading System)

Real-World Example (Grading System Table)

```
create table students(
student_name varchar(30),
marks int
);
```

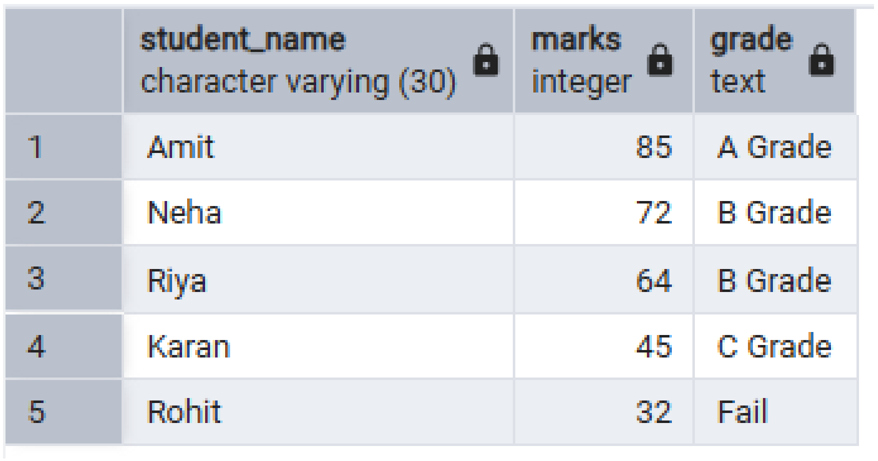
Insert Student Data

```
insert into students values
('Amit',85), ('Neha',72), ('Riya',64), ('Karan',45), ('Rohit',32);
```



	student_name character varying (30)	marks integer
1	Amit	85
2	Neha	72
3	Riya	64
4	Karan	45
5	Rohit	32

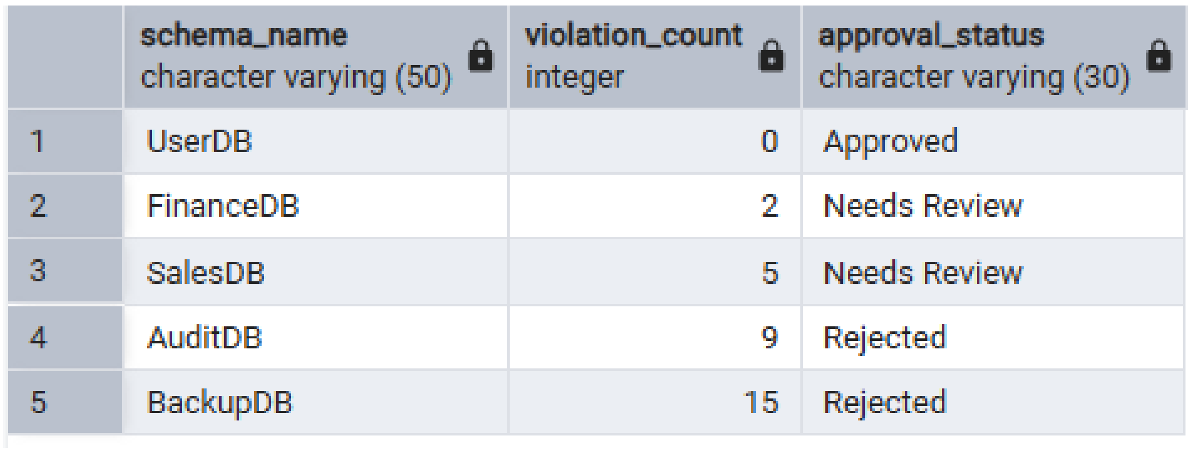
```
select student_name, marks,
case
 when marks >= 80 then 'A Grade'
 when marks >= 60 then 'B Grade'
 when marks >= 40 then 'C Grade'
 else 'Fail'
end as grade
from students;
```



	student_name character varying (30)	marks integer	grade text
1	Amit	85	A Grade
2	Neha	72	B Grade
3	Riya	64	B Grade
4	Karan	45	C Grade
5	Rohit	32	Fail

Step 5: Using CASE for Custom Sorting

```
select schema_name, violation_count, approval_status
from schema_audit
order by
case
 when violation_count = 0 then 1
 when violation_count between 1 and 3 then 2
 when violation_count between 4 and 7 then 3
 else 4
end;
```



	schema_name character varying (50)	violation_count integer	approval_status character varying (30)
1	UserDB	0	Approved
2	FinanceDB	2	Needs Review
3	SalesDB	5	Needs Review
4	AuditDB	9	Rejected
5	BackupDB	15	Rejected

6. I/O Analysis (Input / Output)

Input:

• Schema violation data inserted into the schema_audit table

• CASE expressions for classification and decision making

• ALTER and UPDATE commands with conditional logic

• PL/pgSQL DO block using IF–ELSE conditions

• Student records and grading queries

• Custom sorting queries using CASE

Output:

• Schemas classified into No, Minor, Moderate, and Critical violations

• Automatic approval status generated based on violation count

• Conditional system messages displayed using IF–ELSE logic

• Students categorized into grades based on marks

• Priority-based sorted records displayed

• Correct execution of procedural and conditional SQL logic

Screenshots of execution and obtained results are attached.

7. Learning Outcomes

After completing this experiment, the student has:

• Understood the use of conditional logic in PostgreSQL using CASE expressions.

• Learned to implement decision-making rules directly inside SQL queries.

• Gained hands-on experience with IF–ELSE constructs using PL/pgSQL.

• Developed the ability to classify and validate data based on multiple conditions.

• Practiced real-world scenarios such as compliance checking and grading systems.

• Improved logical thinking skills required for backend development and interviews.

• Built confidence in writing rule-based and analytical SQL queries.