| | |
|---|---|
| **Branch:** MCA (Data Science) Kargil | **Semester:** 2nd |
| **Student Name:** Chaitanya Sharma | **UID:** 25MCD10056 |
| **Subject Name**: Technical Training - I Lab | **Subject Code**: 25CAP-652 |
| **Section/Group:** 25MCD KAR-1 | **Date of Performance:** 27-02-2026 |

# Experiment 6

1. **Experiment Aim:** Learn how to create, query, and manage views in SQL to simplify database queries and provide a layer of abstraction for end-users.

   **(Company Tags: Amazon, Zoho, ServiceNow)**

2. **Tools used:** PostgreSQL

3. **Objectives:**

   - **Data Abstraction:** To understand how to hide complex table joins and calculations behind a simple virtual table interface.
   - **Enhanced Security:** To learn how to restrict user access to sensitive columns by providing views instead of direct table access.

   - **Query Simplification:** To master the creation of views that pre-join multiple tables, making reporting easier for non-technical users.
   - **View Management:** To understand the syntax for creating, altering, and dropping views, as well as the naming conventions required for efficient data access.

- **Theory:** A **View** is essentially a virtual table based on the result-set of an SQL statement. It does not contain data of its own but dynamically pulls data from the underlying "base tables".

i. **Simple Views:** Created from a single table without any aggregate functions or grouping. These are often updatable.

ii. **Complex Views:** Created from multiple tables using JOINs, or including GROUP BY and aggregate functions. These provide a consolidated summary of the database.

iii. **Security Layer:** In enterprise environments, views are used to grant permissions on specific subsets of data. For example, a "SalaryView" might exclude the "Employee_SSN" or "Home_Address" columns for privacy.

iv. **Benefits:** They simplify the user experience, ensure data consistency across reports, and reduce the risk of accidental data modification by providing read-only abstractions.

**4. Experiment Steps:**

Step 1: Creating a Simple View for Data Filtering

**Query:**

```
-- Query 1


-- Create Student table
CREATE TABLE IF NOT EXISTS Student (
    StudentID INT PRIMARY KEY,
    StudentName VARCHAR(50),
    Marks INT,
    EnrollmentStatus VARCHAR(20)
```

```
);

-- Insert sample data
INSERT INTO Student VALUES (201, 'Arjun', 78,
'Enrolled')
ON CONFLICT (StudentID) DO NOTHING;

INSERT INTO Student VALUES (202, 'Sneha', 85,
'Dropped')
ON CONFLICT (StudentID) DO NOTHING;

INSERT INTO Student VALUES (203, 'Ritika', 91,
'Enrolled')
ON CONFLICT (StudentID) DO NOTHING;

-- Create View to show only Enrolled Students
CREATE OR REPLACE VIEW EnrolledStudents AS
SELECT StudentID, StudentName, Marks
FROM Student
WHERE EnrollmentStatus = 'Enrolled';
```

```
-- Query the View

SELECT * FROM EnrolledStudents;SELECT *
```

- **Output:**

| | studentid integer | studentname character varying (50) | marks integer |
|---|---|---|---|
| 1 | 201 | Arjun | 78 |
| 2 | 203 | Ritika | 91 |

*Step 2: Creating a View for Joining Multiple Tables*

**Query:**

```
-- Query 2

-- Create Course table

CREATE TABLE IF NOT EXISTS Course (

    CourseID INT PRIMARY KEY,

    CourseName VARCHAR(50)

);

-- Insert sample courses

INSERT INTO Course VALUES (501, 'Computer Science')

ON CONFLICT (CourseID) DO NOTHING;

INSERT INTO Course VALUES (502, 'Mathematics')

ON CONFLICT (CourseID) DO NOTHING;

INSERT INTO Course VALUES (503, 'Physics')
```

```
ON CONFLICT (CourseID) DO NOTHING;

-- Add CourseID column to Student table

ALTER TABLE Student

ADD COLUMN IF NOT EXISTS CourseID INT;

-- Assign courses to students

UPDATE Student SET CourseID = 501 WHERE StudentID = 201;

UPDATE Student SET CourseID = 502 WHERE StudentID = 202;

UPDATE Student SET CourseID = 503 WHERE StudentID = 203;

-- Create View joining Student and Course

CREATE OR REPLACE VIEW StudentCourseView AS

SELECT

    s.StudentID,

    s.StudentName,

    s.Marks,

    c.CourseName

FROM Student s

JOIN Course c

ON s.CourseID = c.CourseID;

-- Query the View

SELECT * FROM StudentCourseView;
```

- **Output:**

| | studentid integer | studentname character varying (50) | marks integer | coursename character varying (50) |
|---|---|---|---|---|
| 1 | 201 | Arjun | 78 | Computer Science |
| 2 | 202 | Sneha | 85 | Mathematics |
| 3 | 203 | Ritika | 91 | Physics |

*Step 3: Advanced Summarization View*

**Query:**

```
-- Query 3
-- Create summarization view showing course statistics
CREATE OR REPLACE VIEW CourseSummaryView AS

SELECT

    c.CourseName,

    COUNT(s.StudentID) AS TotalStudents,

    AVG(s.Marks) AS AverageMarks,

    SUM(s.Marks) AS TotalMarks

FROM Student s

JOIN Course c

ON s.CourseID = c.CourseID

GROUP BY c.CourseName;
```

-- Query the View

SELECT * FROM CourseSummaryView;

**Output:**

| | coursename character varying (50) | totalstudents bigint | averagemarks numeric | totalmarks bigint |
|---|---|---|---|---|
| 1 | Computer Science | 1 | 78.0000000000000000 | 78 |
| 2 | Mathematics | 1 | 85.0000000000000000 | 85 |
| 3 | Physics | 1 | 91.0000000000000000 | 91 |

**5. Learning Outcomes:**

- Created and use simple views to filter and present specific subsets of data.
- Constructed join-based views to combine related tables into meaningful structured output.
- Built aggregate views using COUNT, AVG, and SUM to generate summarized statistics.