**Lab Exercises on SQL Server Views**

**Prerequisites**

Before starting, create a sample database and tables:

CREATE DATABASE LabDB;

USE LabDB;

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

Name VARCHAR(50),

DepartmentID INT,

Salary DECIMAL(10,2)

);

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(50)

);

INSERT INTO Departments VALUES (1, 'HR'), (2, 'IT'), (3, 'Finance');

INSERT INTO Employees VALUES (1, 'Alice', 1, 50000), (2, 'Bob', 2, 70000), (3, 'Charlie', 3, 60000);

**1. Simple View**

**Exercise**

Create a view that displays only employee names and their respective department IDs.

**Solution**

CREATE VIEW EmployeeView AS

SELECT Name, DepartmentID FROM Employees;

**Test the View**

SELECT \* FROM EmployeeView;

**2. View with JOIN**

**Exercise**

Create a view that displays employee names along with their department names.

**Solution**

CREATE VIEW EmployeeDepartmentView AS

SELECT e.Name, d.DepartmentName

FROM Employees e

JOIN Departments d ON e.DepartmentID = d.DepartmentID;

**Test the View**

SELECT \* FROM EmployeeDepartmentView;

**3. View with Aggregation**

**Exercise**

Create a view that displays the total salary paid in each department.

**Solution**

CREATE VIEW DepartmentSalaryView AS

SELECT d.DepartmentName, SUM(e.Salary) AS TotalSalary

FROM Employees e

JOIN Departments d ON e.DepartmentID = d.DepartmentID

GROUP BY d.DepartmentName;

**Test the View**

SELECT \* FROM DepartmentSalaryView;

**4. View with Computed Columns**

**Exercise**

Create a view that includes an additional computed column, which calculates the annual salary of each employee.

**Solution**

CREATE VIEW EmployeeAnnualSalaryView AS

SELECT Name, Salary, (Salary \* 12) AS AnnualSalary

FROM Employees;

**Test the View**

SELECT \* FROM EmployeeAnnualSalaryView;

**5. View with Filtering (WHERE Clause)**

**Exercise**

Create a view that displays employees who earn more than 60000.

**Solution**

CREATE VIEW HighSalaryEmployees AS

SELECT \* FROM Employees

WHERE Salary > 60000;

**Test the View**

SELECT \* FROM HighSalaryEmployees;

**6. Indexed View (Materialized View)**

**Exercise**

Create an indexed view that calculates the total salary per department.

**Solution**

CREATE VIEW IndexedDepartmentSalaryView

WITH SCHEMABINDING

AS

SELECT e.DepartmentID, SUM(e.Salary) AS TotalSalary, COUNT\_BIG(\*) AS EmployeeCount

FROM dbo.Employees e

GROUP BY e.DepartmentID;

-- Create an index to materialize the view

CREATE UNIQUE CLUSTERED INDEX IDX\_DepartmentSalary

ON IndexedDepartmentSalaryView (DepartmentID);

**Test the View**

SELECT \* FROM IndexedDepartmentSalaryView;

**7. Updatable View (WITH CHECK OPTION)**

**Exercise**

Create a view that allows inserting new employees but restricts modifications outside a specific department.

**Solution**

CREATE VIEW ITDepartmentEmployees AS

SELECT \* FROM Employees

WHERE DepartmentID = 2

WITH CHECK OPTION;

**Test the View**

-- Allowed: Inserting into the IT department

INSERT INTO ITDepartmentEmployees VALUES (4, 'David', 2, 80000);

-- Not Allowed: Inserting into another department

INSERT INTO ITDepartmentEmployees VALUES (5, 'Eve', 3, 75000); -- This will fail

**8. Partitioned View (Union of Tables from Different Databases)**

**Exercise**

Create a partitioned view by combining tables from multiple databases (assume Employees data is split across two databases).

**Solution**

-- Assume LabDB1 and LabDB2 contain parts of the Employees table

USE LabDB1;

CREATE TABLE Employees\_Part1 (

EmployeeID INT PRIMARY KEY,

Name VARCHAR(50),

DepartmentID INT,

Salary DECIMAL(10,2)

);

USE LabDB2;

CREATE TABLE Employees\_Part2 (

EmployeeID INT PRIMARY KEY,

Name VARCHAR(50),

DepartmentID INT,

Salary DECIMAL(10,2)

);

-- Create Partitioned View

USE LabDB;

CREATE VIEW AllEmployees AS

SELECT \* FROM LabDB1.dbo.Employees\_Part1

UNION ALL

SELECT \* FROM LabDB2.dbo.Employees\_Part2;

**Test the View**

SELECT \* FROM AllEmployees;

**9. View with INSTEAD OF Trigger**

**Exercise**

Create a view and an **INSTEAD OF INSERT** trigger to prevent direct modification.

**Solution**

CREATE VIEW ReadOnlyEmployees AS

SELECT \* FROM Employees;

CREATE TRIGGER PreventInsertOnView

INSTEAD OF INSERT ON ReadOnlyEmployees

AS

BEGIN

PRINT 'Insert not allowed on this view';

END;

**Test the View**

INSERT INTO ReadOnlyEmployees VALUES (5, 'Eve', 3, 75000); -- This will fail

**10. Encrypted View**

**Exercise**

Create an encrypted view to hide its definition.

**Solution**

CREATE VIEW EncryptedEmployeeView

WITH ENCRYPTION

AS

SELECT Name, Salary FROM Employees;

**Test the View**

SELECT \* FROM EncryptedEmployeeView;

-- Trying to see the definition will fail:

EXEC sp\_helptext 'EncryptedEmployeeView';