Database Systems
Lab 5: View, Trigger, Store Procedure, Function, Cursor

Student Name

December 5, 2025

Contents

1 Introduction

This lab report covers the implementation of Views, Triggers, Stored Procedures, Functions, and Cursors in MySQL. The exercises are based on the COMPANY database schema and a Hotel reservation system.

1.1 Database Schema: COMPANY

The COMPANY database consists of the following tables:

- `EMPLOYEE` - Employee information
- `DEPARTMENT` - Department information
- `DEPT_LOCATIONS` - Department locations
- `PROJECT` - Project information
- `WORKS_ON` - Employee-Project assignments
- `DEPENDENT` - Employee dependents

2 Views

Exercise: Specify the following views in SQL on the COMPANY database schema:

a. A view that has the department name, manager name, and manager salary for every department.

b. A view that has the employee name, supervisor name, and employee salary for each employee who works in the 'Research' department.

c. A view that has the project name, controlling department name, number of employees, and total hours worked per week on the project for each project.

d. A view that has the project name, controlling department name, number of employees, and total hours worked per week on the project for each project with more than two employees working on it.

e. A view (SSN, Full Name of employee, Number of dependents) that includes information about employees who have the number of dependents greater than 2.

f. A view (Full Name of employee, date of birth, gender) for those employees who have their birthdate in July.

g. A view (Name of dependent, SSN of employee, date of birth of dependent) that includes information on all dependents who are less than 18 years old.

2.1 View (a): Department Manager Information

Requirement: A view that has the department name, manager name, and manager salary for every department.

```sql
DROP VIEW IF EXISTS DepartmentManagerInfo;
CREATE VIEW DepartmentManagerInfo AS
SELECT
    d.Dname AS Department_Name,
    CONCAT(e.Fname, ' ', e.Minit, ' ', e.Lname) AS Manager_Name,
    e.Salary AS Manager_Salary
FROM DEPARTMENT d
JOIN EMPLOYEE e ON d.Mgr_ssn = e.Ssn;
```

Explanation: This view joins the DEPARTMENT and EMPLOYEE tables using the manager's SSN to retrieve the department name, manager's full name (concatenated), and the manager's salary.

Test Validation

```sql
-- Query the view
SELECT * FROM DepartmentManagerInfo;
```

Expected Output:

Department_Name	Manager_Name	Manager_Salary
Research	Franklin T Wong	40000.00
Administration	Jennifer S Wallace	43000.00
Headquarters	James E Borg	55000.00

2.2 View (b): Research Department Employees and Supervisors

Requirement: A view that has the employee name, supervisor name, and employee salary for each employee who works in the 'Research' department.

```sql
DROP VIEW IF EXISTS ResearchEmployeeSupervisor;
CREATE VIEW ResearchEmployeeSupervisor AS
SELECT
    CONCAT(e.Fname, ' ', e.Minit, ' ', e.Lname) AS Employee_Name,
    CONCAT(s.Fname, ' ', s.Minit, ' ', s.Lname) AS Supervisor_Name,
    e.Salary AS Employee_Salary
FROM EMPLOYEE e
LEFT JOIN EMPLOYEE s ON e.Super_ssn = s.Ssn
JOIN DEPARTMENT d ON e.Dno = d.Dnumber
WHERE d.Dname = 'Research';
```

Explanation: This view uses a self-join on the EMPLOYEE table to get supervisor information, with a LEFT JOIN to handle employees without supervisors. The WHERE clause filters for the Research department.

Test Validation

```sql
-- Query the view
SELECT * FROM ResearchEmployeeSupervisor;
```

Expected Output:

Employee_Name	Supervisor_Name	Employee_Salary
John B Smith	Franklin T Wong	30000.00
Franklin T Wong	James E Borg	40000.00
Ramesh K Narayan	Franklin T Wong	38000.00
Joyce A English	Franklin T Wong	25000.00
Ahmad V Jabbar	Franklin T Wong	25000.00

2.3 View (c): Project Information

Requirement: A view that has the project name, controlling department name, number of employees, and total hours worked per week on the project for each project.

```sql
DROP VIEW IF EXISTS ProjectInfo;
CREATE VIEW ProjectInfo AS
SELECT
    p.Pname AS Project_Name,
    d.Dname AS Controlling_Department,
    COUNT(w.Essn) AS Number_of_Employees,
    SUM(IFNULL(w.Hours, 0)) AS Total_Hours_Per_Week
FROM PROJECT p
JOIN DEPARTMENT d ON p.Dnum = d.Dnumber
LEFT JOIN WORKS_ON w ON p.Pnumber = w.Pno
GROUP BY p.Pnumber, p.Pname, d.Dname;
```

Explanation: This view joins PROJECT, DEPARTMENT, and WORKS_ON tables, using GROUP BY to aggregate employee counts and total hours per project.

Test Validation

```sql
-- Query the view
SELECT * FROM ProjectInfo;
```

Expected Output:

Project_Name	Ctrl_Dept	Num_Emp	Total_Hours
ProductX	Research	2	52.5
ProductY	Research	3	37.5
ProductZ	Research	2	50.0
Computerization	Administration	3	55.0
Reorganization	Headquarters	3	25.0
Newbenefits	Administration	3	55.0

2.4 View (d): Projects with More Than Two Employees

Requirement: A view that has the project name, controlling department name, number of employees, and total hours worked per week on the project for each project with more than two employees working on it.

```sql
DROP VIEW IF EXISTS ProjectInfoMoreThanTwo;
CREATE VIEW ProjectInfoMoreThanTwo AS
SELECT
    p.Pname AS Project_Name,
    d.Dname AS Controlling_Department,
    COUNT(w.Essn) AS Number_of_Employees,
    SUM(IFNULL(w.Hours, 0)) AS Total_Hours_Per_Week
```

```sql
FROM PROJECT p
JOIN DEPARTMENT d ON p.Dnum = d.Dnumber
LEFT JOIN WORKS_ON w ON p.Pnumber = w.Pno
GROUP BY p.Pnumber, p.Pname, d.Dname
HAVING COUNT(w.Essn) > 2;
```

Explanation: Similar to View (c), but with a HAVING clause to filter projects that have more than 2 employees.

Test Validation

```sql
-- Query the view
SELECT * FROM ProjectInfoMoreThanTwo;
```

Expected Output:

Project_Name	Ctrl_Dept	Num_Emp	Total_Hours
ProductY	Research	3	37.5
Computerization	Administration	3	55.0
Reorganization	Headquarters	3	25.0
Newbenefits	Administration	3	55.0

2.5 View (e): Employees with More Than 2 Dependents

Requirement: A view (SSN, Full Name of employee, Number of dependents) that includes information about employees who have the number of dependents greater than 2.

```sql
DROP VIEW IF EXISTS EmployeesWithManyDependents;
CREATE VIEW EmployeesWithManyDependents AS
SELECT
    e.Ssn AS SSN,
    CONCAT(e.Fname, ' ', e.Minit, ' ', e.Lname) AS Full_Name,
    COUNT(dep.Dependent_name) AS Number_of_Dependents
FROM EMPLOYEE e
JOIN DEPENDENT dep ON e.Ssn = dep.Essn
GROUP BY e.Ssn, e.Fname, e.Minit, e.Lname
HAVING COUNT(dep.Dependent_name) > 2;
```

Explanation: This view joins EMPLOYEE and DEPENDENT tables, groups by employee, and filters those with more than 2 dependents using HAVING.

Test Validation

```sql
-- Query the view
SELECT * FROM EmployeesWithManyDependents;
```

Expected Output:

SSN	Full_Name	Number_of_Dependents
333445555	Franklin T Wong	3
123456789	John B Smith	3

2.6 View (f): July Birthday Employees

Requirement: A view (Full Name of employee, date of birth, gender) for those employees who have their birthdate in July.

```sql
DROP VIEW IF EXISTS JulyBirthdayEmployees;
CREATE VIEW JulyBirthdayEmployees AS
SELECT
    CONCAT(e.Fname, ' ', e.Minit, ' ', e.Lname) AS Full_Name,
    e.Bdate AS Date_of_Birth,
    e.Sex AS Gender
FROM EMPLOYEE e
WHERE MONTH(e.Bdate) = 7;
```

Explanation: This view uses the MONTH() function to filter employees born in July (month 7).

Test Validation

```sql
-- Query the view
SELECT * FROM JulyBirthdayEmployees;
```

Expected Output:

Full_Name	Date_of_Birth	Gender
Joyce A English	1972-07-31	F
Alicia J Zelaya	1968-07-19	F

2.7 View (g): Young Dependents (Under 18)

Requirement: A view (Name of dependent, SSN of employee, date of birth of dependent) that includes information on all dependents who are less than 18 years old.

```sql
DROP VIEW IF EXISTS YoungDependents;
CREATE VIEW YoungDependents AS
SELECT
    dep.Dependent_name AS Dependent_Name,
    dep.Essn AS Employee_SSN,
    dep.Bdate AS Dependent_Date_of_Birth
FROM DEPENDENT dep
WHERE TIMESTAMPDIFF(YEAR, dep.Bdate, CURDATE()) < 18;
```

Explanation: This view uses TIMESTAMPDIFF() to calculate the age of dependents and filters those under 18 years old.

Test Validation

```sql
-- Query the view
SELECT * FROM YoungDependents;
```

Expected Output: Empty set (no dependents under 18 in current data).

3 Trigger (a) - Business Rules

Exercise: Create a database trigger for the following situations:

- The supervisor of an employee must be older than the employee.
- The salary of an employee cannot be greater than the salary of his/her supervisor.
- The salary of an employee can only increase.
- When increasing salary of employee, the increasing amount must not be more than 20% of current salary.
- An employee works on at most 4 projects.
- The maximum number of hours an employee can work on all projects per week is 56.
- The location of a project must be one of the locations of its department.
- The salary of a department manager must be higher than the other employees working for that department.
- Only department managers can work less than 5 hours on a project.

3.1 Trigger (a.1): Supervisor Must Be Older

Requirement: The supervisor of an employee must be older than the employee.

```sql
DROP TRIGGER IF EXISTS trg_supervisor_older_insert;
DELIMITER //
CREATE TRIGGER trg_supervisor_older_insert
BEFORE INSERT ON EMPLOYEE
FOR EACH ROW
BEGIN
    DECLARE supervisor_bdate DATE;
    IF NEW.Super_ssn IS NOT NULL THEN
        SELECT Bdate INTO supervisor_bdate
        FROM EMPLOYEE WHERE Ssn = NEW.Super_ssn;
        IF supervisor_bdate IS NOT NULL AND NEW.Bdate <=
    supervisor_bdate THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Error: Supervisor must be older than
    the employee.';
        END IF;
    END IF;
END //
DELIMITER ;
```

Explanation: This trigger checks the birthdate of the supervisor before inserting an employee. If the supervisor is not older, it raises an error using SIGNAL.

Test Validation

```sql
-- VALID INSERT: Employee born 1990, Supervisor (333445555) born 1955
INSERT INTO EMPLOYEE VALUES
('Test', 'A', 'Valid', '111111110', '1990-01-01', '123 Test St', 'M',
  25000, '333445555', 5);
-- Result: Success

-- INVALID INSERT: Employee born 1940, Supervisor born 1955 (supervisor
   younger)
INSERT INTO EMPLOYEE VALUES
('Test', 'B', 'Invalid', '111111111', '1940-01-01', '123 Test St', 'M',
   25000, '333445555', 5);
-- Result: Error - Supervisor must be older than the employee.
```

3.2 Trigger (a.2): Salary Cannot Exceed Supervisor's Salary

Requirement: The salary of an employee cannot be greater than the salary of his/her supervisor.

```sql
DROP TRIGGER IF EXISTS trg_salary_less_than_supervisor_insert;
DELIMITER //
CREATE TRIGGER trg_salary_less_than_supervisor_insert
BEFORE INSERT ON EMPLOYEE
FOR EACH ROW
BEGIN
    DECLARE supervisor_salary DECIMAL(10, 2);
    IF NEW.Super_ssn IS NOT NULL THEN
        SELECT Salary INTO supervisor_salary
        FROM EMPLOYEE WHERE Ssn = NEW.Super_ssn;
        IF supervisor_salary IS NOT NULL AND NEW.Salary >
    supervisor_salary THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Error: Employee salary cannot be
    greater than supervisor salary.';
        END IF;
    END IF;
END //
DELIMITER ;
```

Test Validation

```sql
-- VALID INSERT: Employee salary 35000, Supervisor salary 40000
INSERT INTO EMPLOYEE VALUES
('Test', 'C', 'Valid', '111111112', '1990-01-01', '123 Test St', 'M',
    35000, '333445555', 5);
-- Result: Success

-- INVALID INSERT: Employee salary 50000 > Supervisor salary 40000
INSERT INTO EMPLOYEE VALUES
('Test', 'D', 'Invalid', '111111113', '1990-01-01', '123 Test St', 'M',
   50000, '333445555', 5);
-- Result: Error - Employee salary cannot be greater than supervisor
    salary.
```

3.3 Trigger (a.3): Salary Can Only Increase

Requirement: The salary of an employee can only increase.

```sql
DROP TRIGGER IF EXISTS trg_salary_only_increase;
DELIMITER //
CREATE TRIGGER trg_salary_only_increase
BEFORE UPDATE ON EMPLOYEE
```

```sql
FOR EACH ROW
BEGIN
    IF NEW.Salary < OLD.Salary THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Error: Employee salary can only increase, not decrease.';
    END IF;
END //
DELIMITER ;
```

Test Validation

```sql
-- VALID UPDATE: Increase salary from 30000 to 32000
UPDATE EMPLOYEE SET Salary = 32000 WHERE Ssn = '123456789';
-- Result: Success

-- INVALID UPDATE: Decrease salary from 32000 to 28000
UPDATE EMPLOYEE SET Salary = 28000 WHERE Ssn = '123456789';
-- Result: Error - Employee salary can only increase, not decrease.
```

3.4 Trigger (a.4): Maximum 20% Salary Increase

Requirement: When increasing salary of employee, the increasing amount must not be more than 20% of current salary.

```sql
DROP TRIGGER IF EXISTS trg_salary_increase_max_20_percent;
DELIMITER //
CREATE TRIGGER trg_salary_increase_max_20_percent
BEFORE UPDATE ON EMPLOYEE
FOR EACH ROW
BEGIN
    IF NEW.Salary > OLD.Salary THEN
        IF (NEW.Salary - OLD.Salary) > (OLD.Salary * 0.20) THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Error: Salary increase cannot exceed 20% of current salary.';
        END IF;
    END IF;
END //
DELIMITER ;
```

Test Validation

```sql
-- VALID UPDATE: Increase salary by 15% (30000 -> 34500)
UPDATE EMPLOYEE SET Salary = 34500 WHERE Ssn = '123456789';
-- Result: Success

-- INVALID UPDATE: Increase salary by 50% (30000 -> 45000)
UPDATE EMPLOYEE SET Salary = 45000 WHERE Ssn = '123456789';
-- Result: Error - Salary increase cannot exceed 20% of current salary.
```

3.5 Trigger (a.5): Maximum 4 Projects Per Employee

Requirement: An employee works on at most 4 projects.

```sql
DROP TRIGGER IF EXISTS trg_max_4_projects_insert;
DELIMITER //
CREATE TRIGGER trg_max_4_projects_insert
BEFORE INSERT ON WORKS_ON
FOR EACH ROW
BEGIN
    DECLARE project_count INT;
    SELECT COUNT(*) INTO project_count FROM WORKS_ON WHERE Essn = NEW.
   Essn;
    IF project_count >= 4 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Error: An employee can work on at most 4
   projects.';
    END IF;
END //
DELIMITER ;
```

Test Validation

```sql
-- Assume employee '123456789' already works on 4 projects
-- INVALID INSERT: Adding 5th project
INSERT INTO WORKS_ON VALUES ('123456789', 99, 10);
-- Result: Error - An employee can work on at most 4 projects.

-- VALID INSERT: Employee with less than 4 projects
INSERT INTO WORKS_ON VALUES ('999887777', 10, 10);
-- Result: Success
```

3.6 Trigger (a.6): Maximum 56 Hours Per Week

Requirement: The maximum number of hours an employee can work on all projects per week is 56.

```sql
DROP TRIGGER IF EXISTS trg_max_56_hours_insert;
DELIMITER //
CREATE TRIGGER trg_max_56_hours_insert
BEFORE INSERT ON WORKS_ON
FOR EACH ROW
BEGIN
    DECLARE total_hours DECIMAL(5, 1);
    SELECT IFNULL(SUM(Hours), 0) INTO total_hours
    FROM WORKS_ON WHERE Essn = NEW.Essn;
    IF (total_hours + IFNULL(NEW.Hours, 0)) > 56 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Error: Total hours per week cannot exceed
   56.';
    END IF;
END //
DELIMITER ;
```

Test Validation

```sql
-- Assume employee total hours = 40
-- VALID INSERT: Adding 15 hours (total = 55)
INSERT INTO WORKS_ON VALUES ('123456789', 20, 15);
-- Result: Success

-- INVALID INSERT: Adding 20 hours would exceed 56
INSERT INTO WORKS_ON VALUES ('123456789', 30, 20);
-- Result: Error - Total hours per week cannot exceed 56.
```

3.7 Trigger (a.7): Project Location Must Match Department Location

Requirement: The location of a project must be one of the locations of its department.

```sql
DROP TRIGGER IF EXISTS trg_project_location_valid_insert;
DELIMITER //
CREATE TRIGGER trg_project_location_valid_insert
BEFORE INSERT ON PROJECT
FOR EACH ROW
BEGIN
    DECLARE location_exists INT;
    SELECT COUNT(*) INTO location_exists
    FROM DEPT_LOCATIONS
    WHERE Dnumber = NEW.Dnum AND Dlocation = NEW.Plocation;
    IF location_exists = 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Error: Project location must be one of its
   department locations.';
    END IF;
END //
DELIMITER ;
```

Test Validation

```sql
-- Assume Dept 5 has locations: Bellaire, Sugarland, Houston
-- VALID INSERT: Project in valid department location
INSERT INTO PROJECT VALUES ('TestProject', 99, 'Houston', 5);
-- Result: Success

-- INVALID INSERT: Project in location not belonging to department
INSERT INTO PROJECT VALUES ('BadProject', 100, 'New York', 5);
-- Result: Error - Project location must be one of its department
   locations.
```

3.8 Trigger (a.8): Manager Salary Must Be Highest

Requirement: The salary of a department manager must be higher than the other employees working for that department.

```sql
DROP TRIGGER IF EXISTS trg_manager_salary_highest_insert;
DELIMITER //
CREATE TRIGGER trg_manager_salary_highest_insert
BEFORE INSERT ON EMPLOYEE
```

```sql
FOR EACH ROW
BEGIN
    DECLARE manager_salary DECIMAL(10, 2);
    DECLARE manager_ssn CHAR(9);

    IF NEW.Dno IS NOT NULL THEN
        SELECT Mgr_ssn INTO manager_ssn FROM DEPARTMENT WHERE Dnumber =
    NEW.Dno;
        IF manager_ssn IS NOT NULL AND manager_ssn != NEW.Ssn THEN
            SELECT Salary INTO manager_salary FROM EMPLOYEE WHERE Ssn =
    manager_ssn;
            IF manager_salary IS NOT NULL AND NEW.Salary >=
    manager_salary THEN
                SIGNAL SQLSTATE '45000'
                SET MESSAGE_TEXT = 'Error: Employee salary cannot be
    equal or greater than department manager salary.';
            END IF;
        END IF;
    END IF;
END //
DELIMITER ;
```

Test Validation

```sql
-- Assume Dept 5 manager salary = 40000
-- VALID INSERT: Employee salary 35000 < Manager salary 40000
INSERT INTO EMPLOYEE VALUES
('Test', 'E', 'Valid', '111111114', '1990-01-01', '123 St', 'M', 35000,
    '333445555', 5);
-- Result: Success

-- INVALID INSERT: Employee salary 45000 >= Manager salary 40000
INSERT INTO EMPLOYEE VALUES
('Test', 'F', 'Invalid', '111111115', '1990-01-01', '123 St', 'M',
    45000, '333445555', 5);
-- Result: Error - Employee salary cannot be equal or greater than
    manager salary.
```

3.9 Trigger (a.9): Only Managers Can Work Less Than 5 Hours

Requirement: Only department managers can work less than 5 hours on a project.

```sql
DROP TRIGGER IF EXISTS trg_min_5_hours_non_manager_insert;
DELIMITER //
CREATE TRIGGER trg_min_5_hours_non_manager_insert
BEFORE INSERT ON WORKS_ON
FOR EACH ROW
BEGIN
    DECLARE is_manager INT;
    IF NEW.Hours IS NOT NULL AND NEW.Hours < 5 THEN
        SELECT COUNT(*) INTO is_manager FROM DEPARTMENT WHERE Mgr_ssn =
    NEW.Essn;
        IF is_manager = 0 THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Error: Only department managers can
    work less than 5 hours on a project.';
```

```
        END IF;
    END IF;
END //
DELIMITER ;
```

Test Validation

```
-- VALID INSERT: Manager (333445555) working 3 hours
INSERT INTO WORKS_ON VALUES ('333445555', 10, 3);
-- Result: Success (managers can work < 5 hours)

-- INVALID INSERT: Non-manager working 3 hours
INSERT INTO WORKS_ON VALUES ('123456789', 20, 3);
-- Result: Error - Only department managers can work less than 5 hours.

-- VALID INSERT: Non-manager working 10 hours
INSERT INTO WORKS_ON VALUES ('123456789', 20, 10);
-- Result: Success
```

4 Task (b) - Num_of_Emp Derived Attribute

Exercise: Alter table Department to add the attribute Num_of_Emp that stores the number of employees working for each department. This attribute is a derived attribute from Employee.DNO and its value must be automatically calculated.

Solution:

```
-- Add the column
ALTER TABLE DEPARTMENT ADD COLUMN Num_of_Emp INT DEFAULT 0;

-- Initialize the column with current counts
UPDATE DEPARTMENT d
SET Num_of_Emp = (SELECT COUNT(*) FROM EMPLOYEE e WHERE e.Dno = d.
    Dnumber);

-- Trigger for INSERT
DROP TRIGGER IF EXISTS trg_update_num_emp_insert;
DELIMITER //
CREATE TRIGGER trg_update_num_emp_insert
AFTER INSERT ON EMPLOYEE
FOR EACH ROW
BEGIN
    IF NEW.Dno IS NOT NULL THEN
        UPDATE DEPARTMENT
        SET Num_of_Emp = Num_of_Emp + 1
        WHERE Dnumber = NEW.Dno;
    END IF;
END //
DELIMITER ;

-- Trigger for DELETE
DROP TRIGGER IF EXISTS trg_update_num_emp_delete;
DELIMITER //
CREATE TRIGGER trg_update_num_emp_delete
AFTER DELETE ON EMPLOYEE
FOR EACH ROW
```

```
BEGIN
    IF OLD.Dno IS NOT NULL THEN
        UPDATE DEPARTMENT
        SET Num_of_Emp = Num_of_Emp - 1
        WHERE Dnumber = OLD.Dno;
    END IF;
END //
DELIMITER ;

-- Trigger for UPDATE
DROP TRIGGER IF EXISTS trg_update_num_emp_update;
DELIMITER //
CREATE TRIGGER trg_update_num_emp_update
AFTER UPDATE ON EMPLOYEE
FOR EACH ROW
BEGIN
    IF OLD.Dno IS NOT NULL AND (NEW.Dno IS NULL OR OLD.Dno != NEW.Dno)
   THEN
        UPDATE DEPARTMENT SET Num_of_Emp = Num_of_Emp - 1 WHERE Dnumber
    = OLD.Dno;
    END IF;
    IF NEW.Dno IS NOT NULL AND (OLD.Dno IS NULL OR OLD.Dno != NEW.Dno)
   THEN
        UPDATE DEPARTMENT SET Num_of_Emp = Num_of_Emp + 1 WHERE Dnumber
    = NEW.Dno;
    END IF;
END //
DELIMITER ;
```

Test Validation

```
-- Check current department counts
SELECT Dnumber, Dname, Num_of_Emp FROM DEPARTMENT;
```

Expected Output:

Dnumber	Dname	Num_of_Emp
1	Headquarters	1
4	Administration	2
5	Research	5

```
-- Test INSERT: Add new employee to Dept 5
INSERT INTO EMPLOYEE VALUES ('New', 'N', 'Emp', '999999999', '
   1990-01-01',
                                 '123 St', 'M', 25000, '333445555', 5);
SELECT Dnumber, Num_of_Emp FROM DEPARTMENT WHERE Dnumber = 5;
-- Result: Num_of_Emp = 6 (incremented from 5)

-- Test DELETE: Remove the employee
DELETE FROM EMPLOYEE WHERE Ssn = '999999999';
SELECT Dnumber, Num_of_Emp FROM DEPARTMENT WHERE Dnumber = 5;
-- Result: Num_of_Emp = 5 (decremented back)
```

5 Function (c) - Get Total Projects

Exercise: Write a function that returns the total number of projects when given an employee's ID.

- **Input:** employee ID
- **Output:** total number of projects

Solution:

```sql
DROP FUNCTION IF EXISTS GetTotalProjectsForEmployee;
DELIMITER //
CREATE FUNCTION GetTotalProjectsForEmployee(emp_ssn CHAR(9))
RETURNS INT
DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE total_projects INT;

    SELECT COUNT(*) INTO total_projects
    FROM WORKS_ON
    WHERE Essn = emp_ssn;

    RETURN total_projects;
END //
DELIMITER ;

-- Example usage:
SELECT GetTotalProjectsForEmployee('123456789') AS Total_Projects;
SELECT GetTotalProjectsForEmployee('333445555') AS Total_Projects;
```

Explanation: This function takes an employee SSN as input and returns the count of projects that employee works on from the WORKS_ON table.

Test Validation

```sql
-- Call function for employee '123456789'
SELECT GetTotalProjectsForEmployee('123456789') AS Total_Projects;
```

Expected Output:

Total_Projects
2

```sql
-- Call function for employee '333445555'
SELECT GetTotalProjectsForEmployee('333445555') AS Total_Projects;
```

Expected Output:

Total_Projects
2

```sql
-- List all employees with their project counts
SELECT Ssn, CONCAT(Fname, ' ', Lname) AS Name,
       GetTotalProjectsForEmployee(Ssn) AS Projects
FROM EMPLOYEE ORDER BY Projects DESC;
```

Expected Output:

Ssn	Name	Projects
123456789	John Smith	2
333445555	Franklin Wong	2
999887777	Alicia Zelaya	2
987654321	Jennifer Wallace	2
666884444	Ramesh Narayan	3
453453453	Joyce English	2
987987987	Ahmad Jabbar	2
888665555	James Borg	0

6 Procedure (d) - Print Employee Details

Exercise: Create a stored procedure that prints SSN, Full name, Department name, and annual salary of all employees.

Solution:

```sql
DROP PROCEDURE IF EXISTS PrintEmployeeDetails;
DELIMITER //
CREATE PROCEDURE PrintEmployeeDetails()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE v_ssn CHAR(9);
    DECLARE v_fullname VARCHAR(50);
    DECLARE v_dname VARCHAR(25);
    DECLARE v_annual_salary DECIMAL(12, 2);

    DECLARE emp_cursor CURSOR FOR
        SELECT
            e.Ssn,
            CONCAT(e.Fname, ' ', e.Minit, ' ', e.Lname) AS Full_Name,
            d.Dname,
            e.Salary * 12 AS Annual_Salary
        FROM EMPLOYEE e
        LEFT JOIN DEPARTMENT d ON e.Dno = d.Dnumber;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    DROP TEMPORARY TABLE IF EXISTS temp_employee_details;
    CREATE TEMPORARY TABLE temp_employee_details (
        SSN CHAR(9),
        Full_Name VARCHAR(50),
        Department_Name VARCHAR(25),
        Annual_Salary DECIMAL(12, 2)
    );

    OPEN emp_cursor;

    read_loop: LOOP
        FETCH emp_cursor INTO v_ssn, v_fullname, v_dname,
   v_annual_salary;
        IF done THEN
            LEAVE read_loop;
        END IF;
        INSERT INTO temp_employee_details
        VALUES (v_ssn, v_fullname, v_dname, v_annual_salary);
    END LOOP;

    CLOSE emp_cursor;
    SELECT * FROM temp_employee_details;
    DROP TEMPORARY TABLE IF EXISTS temp_employee_details;
END //
DELIMITER ;
```

Test Validation

```
-- Call the procedure
CALL PrintEmployeeDetails();
```

Expected Output:

SSN	Full_Name	Dept_Name	Annual_Salary
123456789	John B Smith	Research	360000.00
333445555	Franklin T Wong	Research	480000.00
999887777	Alicia J Zelaya	Administration	300000.00
987654321	Jennifer S Wallace	Administration	516000.00
666884444	Ramesh K Narayan	Research	456000.00
453453453	Joyce A English	Research	300000.00
987987987	Ahmad V Jabbar	Research	300000.00
888665555	James E Borg	Headquarters	660000.00

7 Trigger (e) - Salary Log

Exercise: Write a trigger that logs any changes in case the new salary is greater than 50000 updated or inserted into our database.

Hint:

- Create a LOG table (SSN, CONTENT, DATE)
- E.g., ('123456789', 'SALARY UPDATE FROM 30000 TO 70000', '06-NOV-2021')

Solution:

```
-- Create the LOG table
DROP TABLE IF EXISTS SALARY_LOG;
CREATE TABLE SALARY_LOG (
    Log_id INT AUTO_INCREMENT PRIMARY KEY,
    SSN CHAR(9),
    Content VARCHAR(255),
    Log_Date DATE
);

-- Trigger for INSERT
DROP TRIGGER IF EXISTS trg_log_salary_insert;
DELIMITER //
CREATE TRIGGER trg_log_salary_insert
AFTER INSERT ON EMPLOYEE
FOR EACH ROW
BEGIN
    IF NEW.Salary > 50000 THEN
```

```sql
        INSERT INTO SALARY_LOG (SSN, Content, Log_Date)
        VALUES (NEW.Ssn, CONCAT('SALARY INSERT: ', NEW.Salary), CURDATE
    ());
    END IF;
END //
DELIMITER ;

-- Trigger for UPDATE
DROP TRIGGER IF EXISTS trg_log_salary_update;
DELIMITER //
CREATE TRIGGER trg_log_salary_update
AFTER UPDATE ON EMPLOYEE
FOR EACH ROW
BEGIN
    IF NEW.Salary > 50000 THEN
        INSERT INTO SALARY_LOG (SSN, Content, Log_Date)
        VALUES (NEW.Ssn, CONCAT('SALARY UPDATE FROM ', OLD.Salary, ' TO
    ', NEW.Salary), CURDATE());
    END IF;
END //
DELIMITER ;
```

Test Validation

```sql
-- Test INSERT with high salary (> 50000)
INSERT INTO EMPLOYEE VALUES ('High', 'H', 'Earner', '888888888', '
    1980-01-01',
                          '123 St', 'M', 55000, NULL, 1);

-- Check the log
SELECT * FROM SALARY_LOG;
```

Expected Output:

Log_id	SSN	Content	Log_Date
1	888888888	SALARY INSERT: 55000.00	2024-12-05

```sql
-- Test UPDATE with high salary
UPDATE EMPLOYEE SET Salary = 60000 WHERE Ssn = '888888888';
SELECT * FROM SALARY_LOG ORDER BY Log_id DESC LIMIT 1;
```

Expected Output:

Log_id	SSN	Content	Log_Date
2	888888888	SALARY UPDATE: 55000 TO 60000	2024-12-05

8 Procedure (f) - Employee Salary Levels

Exercise: Write a stored procedure that prints out the level of salary for each employee.

Rules:

- if (salary < 20000) then “level C”
- if (salary between 20000 and 50000) then “level B”
- if (salary > 50000) then “level A”

Example Output:

```
123456789, John B Smith, level B
333445555, Franklin T Wong, level B
...
```

Solution:

```sql
DROP PROCEDURE IF EXISTS PrintEmployeeSalaryLevel;
DELIMITER //
CREATE PROCEDURE PrintEmployeeSalaryLevel()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE v_ssn CHAR(9);
    DECLARE v_fullname VARCHAR(50);
    DECLARE v_salary DECIMAL(10, 2);
    DECLARE v_level VARCHAR(10);

    DECLARE emp_cursor CURSOR FOR
        SELECT
            e.Ssn,
            CONCAT(e.Fname, ' ', e.Minit, ' ', e.Lname) AS Full_Name,
            e.Salary
        FROM EMPLOYEE e;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    DROP TEMPORARY TABLE IF EXISTS temp_salary_levels;
    CREATE TEMPORARY TABLE temp_salary_levels (
        SSN CHAR(9),
        Full_Name VARCHAR(50),
        Salary_Level VARCHAR(10)
    );

    OPEN emp_cursor;

    read_loop: LOOP
        FETCH emp_cursor INTO v_ssn, v_fullname, v_salary;
        IF done THEN
            LEAVE read_loop;
        END IF;

        IF v_salary < 20000 THEN
            SET v_level = 'level C';
        ELSEIF v_salary >= 20000 AND v_salary <= 50000 THEN
            SET v_level = 'level B';
        ELSE
            SET v_level = 'level A';
        END IF;

        INSERT INTO temp_salary_levels VALUES (v_ssn, v_fullname,
   v_level);
    END LOOP;

    CLOSE emp_cursor;
    SELECT * FROM temp_salary_levels;
    DROP TEMPORARY TABLE IF EXISTS temp_salary_levels;
END //
DELIMITER ;
```

Test Validation

```
-- Call the procedure
CALL PrintEmployeeSalaryLevel();
```

Expected Output:

SSN	Full_Name	Salary_Level
123456789	John B Smith	level B
333445555	Franklin T Wong	level B
999887777	Alicia J Zelaya	level B
987654321	Jennifer S Wallace	level B
666884444	Ramesh K Narayan	level B
453453453	Joyce A English	level B
987987987	Ahmad V Jabbar	level B
888665555	James E Borg	level A

Note: The salary levels are determined by:

- level C: salary < 20000
- level B: $20000 \leq$ salary ≤ 50000
- level A: salary > 50000

9 Exercise 2 - Hotel Database

Exercise: The following tables form part of a database held in a relational DBMS:

- `Hotel (hotelNo, hotelName, city)`
- `Room (roomNo, hotelNo, type, price, NumAdultMax)`
- `Booking (hotelNo, dateFrom, roomNo, guestNo, dateTo, NumOfAdult)`
- `Guest (guestNo, guestName, guestAddress, TotalAmount)`

Where:

- Hotel contains hotel details and hotelNo is the primary key;
- Room contains room details for each hotel and (roomNo, hotelNo) forms the primary key;
- Booking contains details of bookings; (hotelNo, dateFrom, roomNo) forms the primary key; (hotelNo, roomNo) is the foreign key references to Room(roomNo, hotelNo); guestNo is the foreign key references to Guest(guestNo);
- Guest contains guest details and guestNo is the primary key; TotalAmount stores how much one guest has spent and is a derived attribute.

Now alter the Room, Booking, and Guest tables using the integrity enhancement features of SQL with the following constraints:

a. The price of all double rooms must be greater than \$100.

b. In a hotel, the price of double rooms must be greater than the price of the highest single room.

c. A guest cannot make two bookings with overlapping dates.

d. A guest cannot make a booking with number of adults greater than NumAdultMax value of booked room.

e. Automatically calculate the value for TotalAmount column of Guest relation.

f. Create an INSTEAD OF database trigger that will allow data to be inserted into the following view:

```sql
CREATE VIEW LondonHotelRoom AS
SELECT h.hotelNo, hotelName, city, roomNo, type, price
FROM Hotel h, Room r
WHERE h.hotelNo = r.hotelNo AND city = 'London';
```

9.1 Database Schema

```sql
CREATE TABLE Hotel (
    hotelNo INT PRIMARY KEY,
    hotelName VARCHAR(50) NOT NULL,
    city VARCHAR(50) NOT NULL
);

CREATE TABLE Guest (
    guestNo INT PRIMARY KEY,
    guestName VARCHAR(50) NOT NULL,
    guestAddress VARCHAR(100),
    TotalAmount DECIMAL(12, 2) DEFAULT 0
);

CREATE TABLE Room (
    roomNo INT,
    hotelNo INT,
    type VARCHAR(20) NOT NULL,
    price DECIMAL(10, 2) NOT NULL,
    NumAdultMax INT DEFAULT 2,
    PRIMARY KEY (roomNo, hotelNo),
    FOREIGN KEY (hotelNo) REFERENCES Hotel(hotelNo)
);

CREATE TABLE Booking (
    hotelNo INT,
    dateFrom DATE,
    roomNo INT,
    guestNo INT,
    dateTo DATE,
    NumOfAdult INT DEFAULT 1,
    PRIMARY KEY (hotelNo, dateFrom, roomNo),
    FOREIGN KEY (hotelNo, roomNo) REFERENCES Room(roomNo, hotelNo),
    FOREIGN KEY (guestNo) REFERENCES Guest(guestNo)
);
```

9.2 Constraint (a): Double Room Price > $100

```sql
ALTER TABLE Room
ADD CONSTRAINT chk_double_room_price
CHECK (type != 'double' OR price > 100);

-- Alternative trigger implementation:
DROP TRIGGER IF EXISTS trg_double_room_price_insert;
DELIMITER //
CREATE TRIGGER trg_double_room_price_insert
BEFORE INSERT ON Room
FOR EACH ROW
BEGIN
    IF NEW.type = 'double' AND NEW.price <= 100 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Error: Price of double rooms must be
    greater than $100.';
    END IF;
END //
DELIMITER ;
```

Test Validation

```sql
-- Setup test data
INSERT INTO Hotel VALUES (1, 'Test Hotel', 'London');

-- VALID INSERT: Double room with price > 100
INSERT INTO Room VALUES (101, 1, 'double', 150.00, 2);
-- Result: Success

-- INVALID INSERT: Double room with price <= 100
INSERT INTO Room VALUES (102, 1, 'double', 80.00, 2);
-- Result: Error - Price of double rooms must be greater than $100.

-- VALID INSERT: Single room with any price
INSERT INTO Room VALUES (103, 1, 'single', 50.00, 1);
-- Result: Success
```

9.3 Constraint (b): Double Room > Highest Single Room

```sql
DROP TRIGGER IF EXISTS trg_double_greater_single_insert;
DELIMITER //
CREATE TRIGGER trg_double_greater_single_insert
BEFORE INSERT ON Room
FOR EACH ROW
BEGIN
    DECLARE max_single_price DECIMAL(10, 2);

    IF NEW.type = 'double' THEN
        SELECT IFNULL(MAX(price), 0) INTO max_single_price
        FROM Room WHERE hotelNo = NEW.hotelNo AND type = 'single';

        IF NEW.price <= max_single_price THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Error: Double room price must be
    greater than highest single room price.';
```

```
        END IF;
    END IF;
END //
DELIMITER ;
```

Test Validation

```
-- Assume hotel 1 has single room with price 80
INSERT INTO Room VALUES (201, 1, 'single', 80.00, 1);

-- VALID INSERT: Double room price > highest single (80)
INSERT INTO Room VALUES (202, 1, 'double', 120.00, 2);
-- Result: Success

-- INVALID INSERT: Double room price <= highest single (80)
INSERT INTO Room VALUES (203, 1, 'double', 75.00, 2);
-- Result: Error - Double room price must be greater than highest
   single room price.
```

9.4 Constraint (c): No Overlapping Bookings

```
DROP TRIGGER IF EXISTS trg_no_overlapping_bookings_insert;
DELIMITER //
CREATE TRIGGER trg_no_overlapping_bookings_insert
BEFORE INSERT ON Booking
FOR EACH ROW
BEGIN
    DECLARE overlap_count INT;

    SELECT COUNT(*) INTO overlap_count
    FROM Booking
    WHERE guestNo = NEW.guestNo
      AND NOT (NEW.dateTo <= dateFrom OR NEW.dateFrom >= dateTo);

    IF overlap_count > 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Error: Guest cannot have overlapping
   bookings.';
    END IF;
END //
DELIMITER ;
```

Test Validation

```
-- Setup test data
INSERT INTO Guest VALUES (1, 'John Doe', '123 Main St', 0);

-- First booking: Jan 1-5
INSERT INTO Booking VALUES (1, '2024-01-01', 101, 1, '2024-01-05', 2);
-- Result: Success

-- VALID INSERT: Non-overlapping (Jan 10-15)
INSERT INTO Booking VALUES (1, '2024-01-10', 102, 1, '2024-01-15', 1);
-- Result: Success
```

```sql

-- INVALID INSERT: Overlapping with first booking (Jan 3-8)
INSERT INTO Booking VALUES (1, '2024-01-03', 103, 1, '2024-01-08', 1);
-- Result: Error - Guest cannot have overlapping bookings.
```

9.5 Constraint (d): NumOfAdult <= NumAdultMax

```sql
DROP TRIGGER IF EXISTS trg_check_num_adults_insert;
DELIMITER //
CREATE TRIGGER trg_check_num_adults_insert
BEFORE INSERT ON Booking
FOR EACH ROW
BEGIN
    DECLARE max_adults INT;

    SELECT NumAdultMax INTO max_adults
    FROM Room WHERE roomNo = NEW.roomNo AND hotelNo = NEW.hotelNo;

    IF NEW.NumOfAdult > max_adults THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Error: Number of adults exceeds room
    capacity.';
    END IF;
END //
DELIMITER ;
```

Test Validation

```sql
-- Assume Room 101 has NumAdultMax = 2
-- VALID INSERT: 2 adults in room with max 2
INSERT INTO Booking VALUES (1, '2024-02-01', 101, 1, '2024-02-05', 2);
-- Result: Success

-- INVALID INSERT: 4 adults in room with max 2
INSERT INTO Booking VALUES (1, '2024-03-01', 101, 1, '2024-03-05', 4);
-- Result: Error - Number of adults exceeds room capacity.
```

9.6 Constraint (e): Auto-Calculate TotalAmount

```sql
DROP TRIGGER IF EXISTS trg_update_total_amount_insert;
DELIMITER //
CREATE TRIGGER trg_update_total_amount_insert
AFTER INSERT ON Booking
FOR EACH ROW
BEGIN
    DECLARE room_price DECIMAL(10, 2);
    DECLARE num_days INT;
    DECLARE booking_cost DECIMAL(12, 2);

    SELECT price INTO room_price
    FROM Room WHERE roomNo = NEW.roomNo AND hotelNo = NEW.hotelNo;

    SET num_days = DATEDIFF(NEW.dateTo, NEW.dateFrom);
    SET booking_cost = room_price * num_days;

    UPDATE Guest
    SET TotalAmount = TotalAmount + booking_cost
    WHERE guestNo = NEW.guestNo;
END //
DELIMITER ;
```

Test Validation

```sql
-- Check guest's TotalAmount before booking
SELECT guestNo, guestName, TotalAmount FROM Guest WHERE guestNo = 1;
```

Output Before:

guestNo	guestName	TotalAmount
1	John Doe	0.00

```sql
-- Insert booking: Room price = 150, 5 days = 750
INSERT INTO Booking VALUES (1, '2024-04-01', 101, 1, '2024-04-06', 2);

-- Check guest's TotalAmount after booking
SELECT guestNo, guestName, TotalAmount FROM Guest WHERE guestNo = 1;
```

Output After:

guestNo	guestName	TotalAmount
1	John Doe	750.00

9.7 Constraint (f): INSTEAD OF Trigger for LondonHotelRoom View

Note: INSTEAD OF triggers are supported in SQL Server, PostgreSQL, and Oracle, but NOT in MySQL. Below are implementations for different database systems.

The View Definition

```sql
DROP VIEW IF EXISTS LondonHotelRoom;
CREATE VIEW LondonHotelRoom AS
SELECT h.hotelNo, hotelName, city, roomNo, type, price
FROM Hotel h, Room r
WHERE h.hotelNo = r.hotelNo AND city = 'London';
```

SQL Server Syntax

```sql
CREATE TRIGGER trg_instead_of_insert_LondonHotelRoom
ON LondonHotelRoom
INSTEAD OF INSERT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @hotelNo INT, @hotelName VARCHAR(50), @roomNo INT,
            @type VARCHAR(20), @price DECIMAL(10,2);

    -- Get values from the inserted pseudo-table
    SELECT @hotelNo = hotelNo, @hotelName = hotelName,
           @roomNo = roomNo, @type = type, @price = price
    FROM inserted;

    -- Check if hotel exists
    IF NOT EXISTS (SELECT 1 FROM Hotel WHERE hotelNo = @hotelNo)
    BEGIN
        -- Insert new hotel with city = 'London'
        INSERT INTO Hotel (hotelNo, hotelName, city)
        VALUES (@hotelNo, @hotelName, 'London');
    END
    ELSE
    BEGIN
        -- Update hotel name if exists
        UPDATE Hotel SET hotelName = @hotelName WHERE hotelNo =
    @hotelNo;
    END

    -- Insert the room
    INSERT INTO Room (roomNo, hotelNo, type, price)
    VALUES (@roomNo, @hotelNo, @type, @price);
END;
```

PostgreSQL Syntax

```
-- First create the trigger function
CREATE OR REPLACE FUNCTION fn_instead_of_insert_LondonHotelRoom()
RETURNS TRIGGER AS $$
BEGIN
    -- Check if hotel exists
    IF NOT EXISTS (SELECT 1 FROM Hotel WHERE hotelNo = NEW.hotelNo)
    THEN
        INSERT INTO Hotel (hotelNo, hotelName, city)
        VALUES (NEW.hotelNo, NEW.hotelName, 'London');
    ELSE
        UPDATE Hotel SET hotelName = NEW.hotelName
        WHERE hotelNo = NEW.hotelNo;
    END IF;

    -- Insert the room
    INSERT INTO Room (roomNo, hotelNo, type, price)
    VALUES (NEW.roomNo, NEW.hotelNo, NEW.type, NEW.price);

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Create the INSTEAD OF trigger on the view
CREATE TRIGGER trg_instead_of_insert_LondonHotelRoom
INSTEAD OF INSERT ON LondonHotelRoom
FOR EACH ROW
EXECUTE FUNCTION fn_instead_of_insert_LondonHotelRoom();
```

Oracle Syntax

```sql
CREATE OR REPLACE TRIGGER trg_instead_of_insert_LondonHotelRoom
INSTEAD OF INSERT ON LondonHotelRoom
FOR EACH ROW
DECLARE
    v_hotel_count INT;
BEGIN
    -- Check if hotel exists
    SELECT COUNT(*) INTO v_hotel_count
    FROM Hotel WHERE hotelNo = :NEW.hotelNo;

    IF v_hotel_count = 0 THEN
        INSERT INTO Hotel (hotelNo, hotelName, city)
        VALUES (:NEW.hotelNo, :NEW.hotelName, 'London');
    ELSE
        UPDATE Hotel SET hotelName = :NEW.hotelName
        WHERE hotelNo = :NEW.hotelNo;
    END IF;

    -- Insert the room
    INSERT INTO Room (roomNo, hotelNo, type, price)
    VALUES (:NEW.roomNo, :NEW.hotelNo, :NEW.type, :NEW.price);
END;
/
```

Explanation: The INSTEAD OF trigger intercepts INSERT operations on the view and redirects them to the underlying base tables (Hotel and Room). When a user inserts into the LondonHotelRoom view:

1. The trigger checks if the hotel already exists
2. If not, it creates a new hotel record with city = 'London'
3. If yes, it updates the hotel name
4. Finally, it inserts the room into the Room table

Test Validation

```sql
-- Insert into the view using INSTEAD OF trigger
INSERT INTO LondonHotelRoom (hotelNo, hotelName, city, roomNo, type,
    price)
VALUES (10, 'Royal Palace Hotel', 'London', 501, 'double', 250.00);

-- Verify data in base tables
SELECT * FROM Hotel WHERE hotelNo = 10;
SELECT * FROM Room WHERE hotelNo = 10;

-- Query the view
SELECT * FROM LondonHotelRoom WHERE hotelNo = 10;
```

Expected Output (Hotel table):

hotelNo	hotelName	city
10	Royal Palace Hotel	London

Expected Output (Room table):

roomNo	hotelNo	type	price
501	10	double	250.00

Expected Output (View):

hotelNo	hotelName	city	roomNo	type	price
10	Royal Palace Hotel	London	501	double	250.00

10 Conclusion

This lab demonstrated the implementation of various database objects in MySQL:

- **Views:** Created 7 views to present data from multiple tables in a simplified manner
- **Triggers:** Implemented business rules and constraints using BEFORE/AFTER triggers
- **Functions:** Created a user-defined function to encapsulate reusable logic
- **Stored Procedures:** Developed procedures using cursors for iterative processing
- **Constraints:** Applied integrity constraints on the Hotel database

These database objects help maintain data integrity, enforce business rules, and provide convenient data access patterns.