

# Database Systems

## Lab 5: View, Trigger, Store Procedure, Function, Cursor

Student Name

December 5, 2025

### Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Database Schema: COMPANY . . . . .	3
<b>2</b>	<b>Part 1: Views</b>	<b>3</b>
2.1	View (a): Department Manager Information . . . . .	3
2.2	View (b): Research Department Employees and Supervisors . . . . .	4
2.3	View (c): Project Information . . . . .	4
2.4	View (d): Projects with More Than Two Employees . . . . .	5
2.5	View (e): Employees with More Than 2 Dependents . . . . .	5
2.6	View (f): July Birthday Employees . . . . .	6
2.7	View (g): Young Dependents (Under 18) . . . . .	7
<b>3</b>	<b>Part 2: Triggers</b>	<b>7</b>
3.1	Trigger (a.1): Supervisor Must Be Older . . . . .	7
3.2	Trigger (a.2): Salary Cannot Exceed Supervisor's Salary . . . . .	8
3.3	Trigger (a.3): Salary Can Only Increase . . . . .	9
3.4	Trigger (a.4): Maximum 20% Salary Increase . . . . .	9
3.5	Trigger (a.5): Maximum 4 Projects Per Employee . . . . .	10
3.6	Trigger (a.6): Maximum 56 Hours Per Week . . . . .	10
3.7	Trigger (a.7): Project Location Must Match Department Location . . . . .	11
3.8	Trigger (a.8): Manager Salary Must Be Highest . . . . .	11
3.9	Trigger (a.9): Only Managers Can Work Less Than 5 Hours . . . . .	12
3.10	Task (b): Num_of_Emp Derived Attribute . . . . .	13
3.11	Trigger (e): Salary Log for High Salaries . . . . .	15
<b>4</b>	<b>Part 3: Functions</b>	<b>16</b>
4.1	Function (c): Get Total Projects for Employee . . . . .	16
<b>5</b>	<b>Part 4: Stored Procedures</b>	<b>17</b>
5.1	Stored Procedure (d): Print Employee Details . . . . .	17
5.2	Stored Procedure (f): Print Employee Salary Levels . . . . .	18

<b>6</b>	<b>Part 5: Exercise 2 - Hotel Database</b>	<b>20</b>
6.1	Database Schema . . . . .	20
6.2	Constraint (a): Double Room Price > \$100 . . . . .	20
6.3	Constraint (b): Double Room > Highest Single Room . . . . .	21
6.4	Constraint (c): No Overlapping Bookings . . . . .	22
6.5	Constraint (d): NumOfAdult <= NumAdultMax . . . . .	22
6.6	Constraint (e): Auto-Calculate TotalAmount . . . . .	23
6.7	Constraint (f): INSTEAD OF Trigger for LondonHotelRoom View . . . .	24
<b>7</b>	<b>Conclusion</b>	<b>25</b>

# 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 Part 1: Views

### 2.1 View (a): Department Manager Information

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

```

1 DROP VIEW IF EXISTS DepartmentManagerInfo;
2 CREATE VIEW DepartmentManagerInfo AS
3 SELECT
4     d.Dname AS Department_Name,
5     CONCAT(e.Fname, ' ', e.Minit, ' ', e.Lname) AS Manager_Name,
6     e.Salary AS Manager_Salary
7 FROM DEPARTMENT d
8 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

```

1 -- Query the view
2 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.

```

1 DROP VIEW IF EXISTS ResearchEmployeeSupervisor;
2 CREATE VIEW ResearchEmployeeSupervisor AS
3 SELECT
4     CONCAT(e.Fname, ' ', e.Minit, ' ', e.Lname) AS Employee_Name,
5     CONCAT(s.Fname, ' ', s.Minit, ' ', s.Lname) AS Supervisor_Name,
6     e.Salary AS Employee_Salary
7 FROM EMPLOYEE e
8 LEFT JOIN EMPLOYEE s ON e.Super_ssn = s.Ssn
9 JOIN DEPARTMENT d ON e.Dno = d.Dnumber
10 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

```

1 -- Query the view
2 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.

```

1 DROP VIEW IF EXISTS ProjectInfo;
2 CREATE VIEW ProjectInfo AS
3 SELECT
4     p.Pname AS Project_Name,
5     d.Dname AS Controlling_Department,
6     COUNT(w.Essn) AS Number_of_Employees,
7     SUM(IFNULL(w.Hours, 0)) AS Total_Hours_Per_Week
8 FROM PROJECT p
9 JOIN DEPARTMENT d ON p.Dnum = d.Dnumber
10 LEFT JOIN WORKS_ON w ON p.Pnumber = w.Pno
11 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

```
1 -- Query the view
2 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.

```
1 DROP VIEW IF EXISTS ProjectInfoMoreThanTwo;
2 CREATE VIEW ProjectInfoMoreThanTwo AS
3 SELECT
4     p.Pname AS Project_Name,
5     d.Dname AS Controlling_Department,
6     COUNT(w.Essn) AS Number_of_Employees,
7     SUM(IFNULL(w.Hours, 0)) AS Total_Hours_Per_Week
8 FROM PROJECT p
9 JOIN DEPARTMENT d ON p.Dnum = d.Dnumber
10 LEFT JOIN WORKS_ON w ON p.Pnumber = w.Pno
11 GROUP BY p.Pnumber, p.Pname, d.Dname
12 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

```
1 -- Query the view
2 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.

```

1 DROP VIEW IF EXISTS EmployeesWithManyDependents;
2 CREATE VIEW EmployeesWithManyDependents AS
3 SELECT
4     e.Ssn AS SSN,
5     CONCAT(e.Fname, ' ', e.Minit, ' ', e.Lname) AS Full_Name,
6     COUNT(dep.Dependent_name) AS Number_of_Dependents
7 FROM EMPLOYEE e
8 JOIN DEPENDENT dep ON e.Ssn = dep.Essn
9 GROUP BY e.Ssn, e.Fname, e.Minit, e.Lname
10 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

```

1 -- Query the view
2 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.

```

1 DROP VIEW IF EXISTS JulyBirthdayEmployees;
2 CREATE VIEW JulyBirthdayEmployees AS
3 SELECT
4     CONCAT(e.Fname, ' ', e.Minit, ' ', e.Lname) AS Full_Name,
5     e.Bdate AS Date_of_Birth,
6     e.Sex AS Gender
7 FROM EMPLOYEE e
8 WHERE MONTH(e.Bdate) = 7;

```

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

### Test Validation

```

1 -- Query the view
2 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.

```
1 DROP VIEW IF EXISTS YoungDependents;
2 CREATE VIEW YoungDependents AS
3 SELECT
4     dep.Dependent_name AS Dependent_Name,
5     dep.Essn AS Employee_SSN,
6     dep.Bdate AS Dependent_Date_of_Birth
7 FROM DEPENDENT dep
8 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

```
1 -- Query the view
2 SELECT * FROM YoungDependents;
```

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

## 3 Part 2: Triggers

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

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

```
1 DROP TRIGGER IF EXISTS trg_supervisor_older_insert;
2 DELIMITER //
3 CREATE TRIGGER trg_supervisor_older_insert
4 BEFORE INSERT ON EMPLOYEE
5 FOR EACH ROW
6 BEGIN
7     DECLARE supervisor_bdate DATE;
8     IF NEW.Super_ssn IS NOT NULL THEN
9         SELECT Bdate INTO supervisor_bdate
10        FROM EMPLOYEE WHERE Ssn = NEW.Super_ssn;
11        IF supervisor_bdate IS NOT NULL AND NEW.Bdate <=
supervisor_bdate THEN
12            SIGNAL SQLSTATE '45000'
13            SET MESSAGE_TEXT = 'Error: Supervisor must be older than
the employee.';
14        END IF;
15    END IF;
16 END //
17 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

```

1 -- VALID INSERT: Employee born 1990, Supervisor (333445555) born 1955
2 INSERT INTO EMPLOYEE VALUES
3 ('Test', 'A', 'Valid', '11111110', '1990-01-01', '123 Test St', 'M',
   25000, '333445555', 5);
4 -- Result: Success
5
6 -- INVALID INSERT: Employee born 1940, Supervisor born 1955 (supervisor
   younger)
7 INSERT INTO EMPLOYEE VALUES
8 ('Test', 'B', 'Invalid', '11111111', '1940-01-01', '123 Test St', 'M',
   25000, '333445555', 5);
9 -- 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.

```

1 DROP TRIGGER IF EXISTS trg_salary_less_than_supervisor_insert;
2 DELIMITER //
3 CREATE TRIGGER trg_salary_less_than_supervisor_insert
4 BEFORE INSERT ON EMPLOYEE
5 FOR EACH ROW
6 BEGIN
7     DECLARE supervisor_salary DECIMAL(10, 2);
8     IF NEW.Super_ssn IS NOT NULL THEN
9         SELECT Salary INTO supervisor_salary
10        FROM EMPLOYEE WHERE Ssn = NEW.Super_ssn;
11        IF supervisor_salary IS NOT NULL AND NEW.Salary >
supervisor_salary THEN
12            SIGNAL SQLSTATE '45000'
13            SET MESSAGE_TEXT = 'Error: Employee salary cannot be
greater than supervisor salary.';
14        END IF;
15    END IF;
16 END //
17 DELIMITER ;

```

#### Test Validation

```

1 -- VALID INSERT: Employee salary 35000, Supervisor salary 40000
2 INSERT INTO EMPLOYEE VALUES
3 ('Test', 'C', 'Valid', '11111112', '1990-01-01', '123 Test St', 'M',
   35000, '333445555', 5);
4 -- Result: Success
5
6 -- INVALID INSERT: Employee salary 50000 > Supervisor salary 40000
7 INSERT INTO EMPLOYEE VALUES
8 ('Test', 'D', 'Invalid', '11111113', '1990-01-01', '123 Test St', 'M',
   50000, '333445555', 5);
9 -- 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.

```

1 DROP TRIGGER IF EXISTS trg_salary_only_increase;
2 DELIMITER //
3 CREATE TRIGGER trg_salary_only_increase
4 BEFORE UPDATE ON EMPLOYEE
5 FOR EACH ROW
6 BEGIN
7     IF NEW.Salary < OLD.Salary THEN
8         SIGNAL SQLSTATE '45000'
9         SET MESSAGE_TEXT = 'Error: Employee salary can only increase,
10         not decrease.';
11     END IF;
12 END //
13 DELIMITER ;

```

#### Test Validation

```

1 -- VALID UPDATE: Increase salary from 30000 to 32000
2 UPDATE EMPLOYEE SET Salary = 32000 WHERE Ssn = '123456789';
3 -- Result: Success
4
5 -- INVALID UPDATE: Decrease salary from 32000 to 28000
6 UPDATE EMPLOYEE SET Salary = 28000 WHERE Ssn = '123456789';
7 -- 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.

```

1 DROP TRIGGER IF EXISTS trg_salary_increase_max_20_percent;
2 DELIMITER //
3 CREATE TRIGGER trg_salary_increase_max_20_percent
4 BEFORE UPDATE ON EMPLOYEE
5 FOR EACH ROW
6 BEGIN
7     IF NEW.Salary > OLD.Salary THEN
8         IF (NEW.Salary - OLD.Salary) > (OLD.Salary * 0.20) THEN
9             SIGNAL SQLSTATE '45000'
10             SET MESSAGE_TEXT = 'Error: Salary increase cannot exceed
11             20% of current salary.';
12         END IF;
13     END IF;
14 END //
15 DELIMITER ;

```

#### Test Validation

```

1 -- VALID UPDATE: Increase salary by 15% (30000 -> 34500)
2 UPDATE EMPLOYEE SET Salary = 34500 WHERE Ssn = '123456789';
3 -- Result: Success
4

```

```

5 -- INVALID UPDATE: Increase salary by 50% (30000 -> 45000)
6 UPDATE EMPLOYEE SET Salary = 45000 WHERE Ssn = '123456789';
7 -- 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.

```

1 DROP TRIGGER IF EXISTS trg_max_4_projects_insert;
2 DELIMITER //
3 CREATE TRIGGER trg_max_4_projects_insert
4 BEFORE INSERT ON WORKS_ON
5 FOR EACH ROW
6 BEGIN
7     DECLARE project_count INT;
8     SELECT COUNT(*) INTO project_count FROM WORKS_ON WHERE Essn = NEW.
      Essn;
9     IF project_count >= 4 THEN
10         SIGNAL SQLSTATE '45000'
11         SET MESSAGE_TEXT = 'Error: An employee can work on at most 4
      projects.';
12     END IF;
13 END //
14 DELIMITER ;

```

#### Test Validation

```

1 -- Assume employee '123456789' already works on 4 projects
2 -- INVALID INSERT: Adding 5th project
3 INSERT INTO WORKS_ON VALUES ('123456789', 99, 10);
4 -- Result: Error - An employee can work on at most 4 projects.
5
6 -- VALID INSERT: Employee with less than 4 projects
7 INSERT INTO WORKS_ON VALUES ('999887777', 10, 10);
8 -- 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.

```

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

```

```

14 END //
15 DELIMITER ;

```

### Test Validation

```

1 -- Assume employee total hours = 40
2 -- VALID INSERT: Adding 15 hours (total = 55)
3 INSERT INTO WORKS_ON VALUES ('123456789', 20, 15);
4 -- Result: Success
5
6 -- INVALID INSERT: Adding 20 hours would exceed 56
7 INSERT INTO WORKS_ON VALUES ('123456789', 30, 20);
8 -- 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.

```

1 DROP TRIGGER IF EXISTS trg_project_location_valid_insert;
2 DELIMITER //
3 CREATE TRIGGER trg_project_location_valid_insert
4 BEFORE INSERT ON PROJECT
5 FOR EACH ROW
6 BEGIN
7     DECLARE location_exists INT;
8     SELECT COUNT(*) INTO location_exists
9     FROM DEPT_LOCATIONS
10    WHERE Dnumber = NEW.Dnum AND Dlocation = NEW.Plocation;
11    IF location_exists = 0 THEN
12        SIGNAL SQLSTATE '45000'
13        SET MESSAGE_TEXT = 'Error: Project location must be one of its
14        department locations.';
15    END IF;
16 END //
17 DELIMITER ;

```

### Test Validation

```

1 -- Assume Dept 5 has locations: Bellaire, Sugarland, Houston
2 -- VALID INSERT: Project in valid department location
3 INSERT INTO PROJECT VALUES ('TestProject', 99, 'Houston', 5);
4 -- Result: Success
5
6 -- INVALID INSERT: Project in location not belonging to department
7 INSERT INTO PROJECT VALUES ('BadProject', 100, 'New York', 5);
8 -- Result: Error - Project location must be one of its department
9   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.

```

1 DROP TRIGGER IF EXISTS trg_manager_salary_highest_insert;
2 DELIMITER //
3 CREATE TRIGGER trg_manager_salary_highest_insert
4 BEFORE INSERT ON EMPLOYEE
5 FOR EACH ROW
6 BEGIN
7     DECLARE manager_salary DECIMAL(10, 2);
8     DECLARE manager_ssn CHAR(9);
9
10    IF NEW.Dno IS NOT NULL THEN
11        SELECT Mgr_ssn INTO manager_ssn FROM DEPARTMENT WHERE Dnumber =
NEW.Dno;
12        IF manager_ssn IS NOT NULL AND manager_ssn != NEW.Ssn THEN
13            SELECT Salary INTO manager_salary FROM EMPLOYEE WHERE Ssn =
manager_ssn;
14            IF manager_salary IS NOT NULL AND NEW.Salary >=
manager_salary THEN
15                SIGNAL SQLSTATE '45000'
16                SET MESSAGE_TEXT = 'Error: Employee salary cannot be
equal or greater than department manager salary.';
17            END IF;
18        END IF;
19    END IF;
20 END //
21 DELIMITER ;

```

## Test Validation

```

1 -- Assume Dept 5 manager salary = 40000
2 -- VALID INSERT: Employee salary 35000 < Manager salary 40000
3 INSERT INTO EMPLOYEE VALUES
4 ('Test', 'E', 'Valid', '11111114', '1990-01-01', '123 St', 'M', 35000,
'333445555', 5);
5 -- Result: Success
6
7 -- INVALID INSERT: Employee salary 45000 >= Manager salary 40000
8 INSERT INTO EMPLOYEE VALUES
9 ('Test', 'F', 'Invalid', '11111115', '1990-01-01', '123 St', 'M',
45000, '333445555', 5);
10 -- 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.

```

1 DROP TRIGGER IF EXISTS trg_min_5_hours_non_manager_insert;
2 DELIMITER //
3 CREATE TRIGGER trg_min_5_hours_non_manager_insert
4 BEFORE INSERT ON WORKS_ON
5 FOR EACH ROW
6 BEGIN
7     DECLARE is_manager INT;
8     IF NEW.Hours IS NOT NULL AND NEW.Hours < 5 THEN
9         SELECT COUNT(*) INTO is_manager FROM DEPARTMENT WHERE Mgr_ssn =
NEW.Essn;

```

```

10         IF is_manager = 0 THEN
11             SIGNAL SQLSTATE '45000'
12             SET MESSAGE_TEXT = 'Error: Only department managers can
work less than 5 hours on a project.';
13         END IF;
14     END IF;
15 END //
16 DELIMITER ;

```

## Test Validation

```

1 -- VALID INSERT: Manager (333445555) working 3 hours
2 INSERT INTO WORKS_ON VALUES ('333445555', 10, 3);
3 -- Result: Success (managers can work < 5 hours)
4
5 -- INVALID INSERT: Non-manager working 3 hours
6 INSERT INTO WORKS_ON VALUES ('123456789', 20, 3);
7 -- Result: Error - Only department managers can work less than 5 hours.
8
9 -- VALID INSERT: Non-manager working 10 hours
10 INSERT INTO WORKS_ON VALUES ('123456789', 20, 10);
11 -- Result: Success

```

## 3.10 Task (b): Num\_of\_Emp Derived Attribute

**Requirement:** 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.

```

1 -- Add the column
2 ALTER TABLE DEPARTMENT ADD COLUMN Num_of_Emp INT DEFAULT 0;
3
4 -- Initialize the column with current counts
5 UPDATE DEPARTMENT d
6 SET Num_of_Emp = (SELECT COUNT(*) FROM EMPLOYEE e WHERE e.Dno = d.
Dnumber);
7
8 -- Trigger for INSERT
9 DROP TRIGGER IF EXISTS trg_update_num_emp_insert;
10 DELIMITER //
11 CREATE TRIGGER trg_update_num_emp_insert
12 AFTER INSERT ON EMPLOYEE
13 FOR EACH ROW
14 BEGIN
15     IF NEW.Dno IS NOT NULL THEN
16         UPDATE DEPARTMENT
17         SET Num_of_Emp = Num_of_Emp + 1
18         WHERE Dnumber = NEW.Dno;
19     END IF;
20 END //
21 DELIMITER ;
22
23 -- Trigger for DELETE
24 DROP TRIGGER IF EXISTS trg_update_num_emp_delete;
25 DELIMITER //
26 CREATE TRIGGER trg_update_num_emp_delete

```

```

27 AFTER DELETE ON EMPLOYEE
28 FOR EACH ROW
29 BEGIN
30     IF OLD.Dno IS NOT NULL THEN
31         UPDATE DEPARTMENT
32         SET Num_of_Emp = Num_of_Emp - 1
33         WHERE Dnumber = OLD.Dno;
34     END IF;
35 END //
36 DELIMITER ;
37
38 -- Trigger for UPDATE
39 DROP TRIGGER IF EXISTS trg_update_num_emp_update;
40 DELIMITER //
41 CREATE TRIGGER trg_update_num_emp_update
42 AFTER UPDATE ON EMPLOYEE
43 FOR EACH ROW
44 BEGIN
45     IF OLD.Dno IS NOT NULL AND (NEW.Dno IS NULL OR OLD.Dno != NEW.Dno)
46     THEN
47         UPDATE DEPARTMENT SET Num_of_Emp = Num_of_Emp - 1 WHERE Dnumber
48         = OLD.Dno;
49     END IF;
50     IF NEW.Dno IS NOT NULL AND (OLD.Dno IS NULL OR OLD.Dno != NEW.Dno)
51     THEN
52         UPDATE DEPARTMENT SET Num_of_Emp = Num_of_Emp + 1 WHERE Dnumber
53         = NEW.Dno;
54     END IF;
55 END //
56 DELIMITER ;

```

## Test Validation

```

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

```

### Expected Output:

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

```

1 -- Test INSERT: Add new employee to Dept 5
2 INSERT INTO EMPLOYEE VALUES ('New', 'N', 'Emp', '999999999', '
3     '1990-01-01',
4     '123 St', 'M', 25000, '333445555', 5);
5 SELECT Dnumber, Num_of_Emp FROM DEPARTMENT WHERE Dnumber = 5;
6 -- Result: Num_of_Emp = 6 (incremented from 5)
7
8 -- Test DELETE: Remove the employee
9 DELETE FROM EMPLOYEE WHERE Ssn = '999999999';
10 SELECT Dnumber, Num_of_Emp FROM DEPARTMENT WHERE Dnumber = 5;
-- Result: Num_of_Emp = 5 (decremented back)

```

### 3.11 Trigger (e): Salary Log for High Salaries

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

```

1  -- Create the LOG table
2  DROP TABLE IF EXISTS SALARY_LOG;
3  CREATE TABLE SALARY_LOG (
4      Log_id INT AUTO_INCREMENT PRIMARY KEY,
5      SSN CHAR(9),
6      Content VARCHAR(255),
7      Log_Date DATE
8  );
9
10 -- Trigger for INSERT
11 DROP TRIGGER IF EXISTS trg_log_salary_insert;
12 DELIMITER //
13 CREATE TRIGGER trg_log_salary_insert
14 AFTER INSERT ON EMPLOYEE
15 FOR EACH ROW
16 BEGIN
17     IF NEW.Salary > 50000 THEN
18         INSERT INTO SALARY_LOG (SSN, Content, Log_Date)
19         VALUES (NEW.Ssn, CONCAT('SALARY INSERT: ', NEW.Salary), CURDATE
20         ());
21     END IF;
22 END //
23 DELIMITER ;
24
25 -- Trigger for UPDATE
26 DROP TRIGGER IF EXISTS trg_log_salary_update;
27 DELIMITER //
28 CREATE TRIGGER trg_log_salary_update
29 AFTER UPDATE ON EMPLOYEE
30 FOR EACH ROW
31 BEGIN
32     IF NEW.Salary > 50000 THEN
33         INSERT INTO SALARY_LOG (SSN, Content, Log_Date)
34         VALUES (NEW.Ssn, CONCAT('SALARY UPDATE FROM ', OLD.Salary, ' TO
35         ', NEW.Salary), CURDATE());
36     END IF;
37 END //
38 DELIMITER ;

```

#### Test Validation

```

1  -- Test INSERT with high salary (> 50000)
2  INSERT INTO EMPLOYEE VALUES ('High', 'H', 'Earner', '888888888', '
3      1980-01-01',
4      '123 St', 'M', 55000, NULL, 1);
5
6  -- Check the log
7  SELECT * FROM SALARY_LOG;

```

Expected Output:

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

```

1 -- Test UPDATE with high salary
2 UPDATE EMPLOYEE SET Salary = 60000 WHERE Ssn = '888888888';
3 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

## 4 Part 3: Functions

### 4.1 Function (c): Get Total Projects for Employee

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

```

1 DROP FUNCTION IF EXISTS GetTotalProjectsForEmployee;
2 DELIMITER //
3 CREATE FUNCTION GetTotalProjectsForEmployee(emp_ssn CHAR(9))
4 RETURNS INT
5 DETERMINISTIC
6 READS SQL DATA
7 BEGIN
8     DECLARE total_projects INT;
9
10    SELECT COUNT(*) INTO total_projects
11    FROM WORKS_ON
12    WHERE Essn = emp_ssn;
13
14    RETURN total_projects;
15 END //
16 DELIMITER ;
17
18 -- Example usage:
19 SELECT GetTotalProjectsForEmployee('123456789') AS Total_Projects;
20 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

```

1 -- Call function for employee '123456789'
2 SELECT GetTotalProjectsForEmployee('123456789') AS Total_Projects;

```

Expected Output:

Total_Projects
2

```

1 -- Call function for employee '333445555'
2 SELECT GetTotalProjectsForEmployee('333445555') AS Total_Projects;

```

Expected Output:

Total_Projects
2



```

1 -- List all employees with their project counts
2 SELECT Ssn, CONCAT(Fname, ' ', Lname) AS Name,
3       GetTotalProjectsForEmployee(Ssn) AS Projects
4 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

## 5 Part 4: Stored Procedures

### 5.1 Stored Procedure (d): Print Employee Details

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

```

1 DROP PROCEDURE IF EXISTS PrintEmployeeDetails;
2 DELIMITER //
3 CREATE PROCEDURE PrintEmployeeDetails()
4 BEGIN
5     DECLARE done INT DEFAULT FALSE;
6     DECLARE v_ssn CHAR(9);
7     DECLARE v_fullname VARCHAR(50);
8     DECLARE v_dname VARCHAR(25);
9     DECLARE v_annual_salary DECIMAL(12, 2);
10
11     DECLARE emp_cursor CURSOR FOR
12     SELECT
13         e.Ssn,
14         CONCAT(e.Fname, ' ', e.Minit, ' ', e.Lname) AS Full_Name,
15         d.Dname,
16         e.Salary * 12 AS Annual_Salary
17     FROM EMPLOYEE e
18     LEFT JOIN DEPARTMENT d ON e.Dno = d.Dnumber;
19
20     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
21
22     DROP TEMPORARY TABLE IF EXISTS temp_employee_details;
23     CREATE TEMPORARY TABLE temp_employee_details (
24         SSN CHAR(9),
25         Full_Name VARCHAR(50),
26         Department_Name VARCHAR(25),
27         Annual_Salary DECIMAL(12, 2)
28     );
29
30     OPEN emp_cursor;
31
32     read_loop: LOOP

```

```

33      FETCH emp_cursor INTO v_ssn, v_fullname, v_dname,
      v_annual_salary;
34      IF done THEN
35          LEAVE read_loop;
36      END IF;
37      INSERT INTO temp_employee_details
38      VALUES (v_ssn, v_fullname, v_dname, v_annual_salary);
39  END LOOP;
40
41  CLOSE emp_cursor;
42  SELECT * FROM temp_employee_details;
43  DROP TEMPORARY TABLE IF EXISTS temp_employee_details;
44 END //
45 DELIMITER ;

```

## Test Validation

```

1  -- Call the procedure
2  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

## 5.2 Stored Procedure (f): Print Employee Salary Levels

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

```

1  DROP PROCEDURE IF EXISTS PrintEmployeeSalaryLevel;
2  DELIMITER //
3  CREATE PROCEDURE PrintEmployeeSalaryLevel()
4  BEGIN
5      DECLARE done INT DEFAULT FALSE;
6      DECLARE v_ssn CHAR(9);
7      DECLARE v_fullname VARCHAR(50);
8      DECLARE v_salary DECIMAL(10, 2);
9      DECLARE v_level VARCHAR(10);
10
11     DECLARE emp_cursor CURSOR FOR
12     SELECT
13         e.Ssn,
14         CONCAT(e.Fname, ' ', e.Minit, ' ', e.Lname) AS Full_Name,
15         e.Salary
16     FROM EMPLOYEE e;
17
18     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
19
20     DROP TEMPORARY TABLE IF EXISTS temp_salary_levels;

```

```

21 CREATE TEMPORARY TABLE temp_salary_levels (
22     SSN CHAR(9),
23     Full_Name VARCHAR(50),
24     Salary_Level VARCHAR(10)
25 );
26
27 OPEN emp_cursor;
28
29 read_loop: LOOP
30     FETCH emp_cursor INTO v_ssn, v_fullname, v_salary;
31     IF done THEN
32         LEAVE read_loop;
33     END IF;
34
35     IF v_salary < 20000 THEN
36         SET v_level = 'level C';
37     ELSEIF v_salary >= 20000 AND v_salary <= 50000 THEN
38         SET v_level = 'level B';
39     ELSE
40         SET v_level = 'level A';
41     END IF;
42
43     INSERT INTO temp_salary_levels VALUES (v_ssn, v_fullname,
44     v_level);
45     END LOOP;
46
47 CLOSE emp_cursor;
48 SELECT * FROM temp_salary_levels;
49 DROP TEMPORARY TABLE IF EXISTS temp_salary_levels;
50 END //
51 DELIMITER ;

```

## Test Validation

```

1 -- Call the procedure
2 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 ≤ salary ≤ 50000
- level A: salary > 50000

## 6 Part 5: Exercise 2 - Hotel Database

### 6.1 Database Schema

```
1 CREATE TABLE Hotel (  
2     hotelNo INT PRIMARY KEY,  
3     hotelName VARCHAR(50) NOT NULL,  
4     city VARCHAR(50) NOT NULL  
5 );  
6  
7 CREATE TABLE Guest (  
8     guestNo INT PRIMARY KEY,  
9     guestName VARCHAR(50) NOT NULL,  
10    guestAddress VARCHAR(100),  
11    TotalAmount DECIMAL(12, 2) DEFAULT 0  
12 );  
13  
14 CREATE TABLE Room (  
15     roomNo INT,  
16     hotelNo INT,  
17     type VARCHAR(20) NOT NULL,  
18     price DECIMAL(10, 2) NOT NULL,  
19     NumAdultMax INT DEFAULT 2,  
20     PRIMARY KEY (roomNo, hotelNo),  
21     FOREIGN KEY (hotelNo) REFERENCES Hotel(hotelNo)  
22 );  
23  
24 CREATE TABLE Booking (  
25     hotelNo INT,  
26     dateFrom DATE,  
27     roomNo INT,  
28     guestNo INT,  
29     dateTo DATE,  
30     NumOfAdult INT DEFAULT 1,  
31     PRIMARY KEY (hotelNo, dateFrom, roomNo),  
32     FOREIGN KEY (hotelNo, roomNo) REFERENCES Room(roomNo, hotelNo),  
33     FOREIGN KEY (guestNo) REFERENCES Guest(guestNo)  
34 );
```

### 6.2 Constraint (a): Double Room Price > \$100

```
1 ALTER TABLE Room  
2 ADD CONSTRAINT chk_double_room_price  
3 CHECK (type != 'double' OR price > 100);  
4  
5 -- Alternative trigger implementation:  
6 DROP TRIGGER IF EXISTS trg_double_room_price_insert;  
7 DELIMITER //  
8 CREATE TRIGGER trg_double_room_price_insert  
9 BEFORE INSERT ON Room  
10 FOR EACH ROW  
11 BEGIN  
12     IF NEW.type = 'double' AND NEW.price <= 100 THEN  
13         SIGNAL SQLSTATE '45000',  
14         SET MESSAGE_TEXT = 'Error: Price of double rooms must be  
greater than $100.';
```

```

15     END IF;
16 END //
17 DELIMITER ;

```

## Test Validation

```

1  -- Setup test data
2  INSERT INTO Hotel VALUES (1, 'Test Hotel', 'London');
3
4  -- VALID INSERT: Double room with price > 100
5  INSERT INTO Room VALUES (101, 1, 'double', 150.00, 2);
6  -- Result: Success
7
8  -- INVALID INSERT: Double room with price <= 100
9  INSERT INTO Room VALUES (102, 1, 'double', 80.00, 2);
10 -- Result: Error - Price of double rooms must be greater than $100.
11
12 -- VALID INSERT: Single room with any price
13 INSERT INTO Room VALUES (103, 1, 'single', 50.00, 1);
14 -- Result: Success

```

## 6.3 Constraint (b): Double Room > Highest Single Room

```

1 DROP TRIGGER IF EXISTS trg_double_greater_single_insert;
2 DELIMITER //
3 CREATE TRIGGER trg_double_greater_single_insert
4 BEFORE INSERT ON Room
5 FOR EACH ROW
6 BEGIN
7     DECLARE max_single_price DECIMAL(10, 2);
8
9     IF NEW.type = 'double' THEN
10         SELECT IFNULL(MAX(price), 0) INTO max_single_price
11         FROM Room WHERE hotelNo = NEW.hotelNo AND type = 'single';
12
13         IF NEW.price <= max_single_price THEN
14             SIGNAL SQLSTATE '45000'
15             SET MESSAGE_TEXT = 'Error: Double room price must be
greater than highest single room price.';
16         END IF;
17     END IF;
18 END //
19 DELIMITER ;

```

## Test Validation

```

1  -- Assume hotel 1 has single room with price 80
2  INSERT INTO Room VALUES (201, 1, 'single', 80.00, 1);
3
4  -- VALID INSERT: Double room price > highest single (80)
5  INSERT INTO Room VALUES (202, 1, 'double', 120.00, 2);
6  -- Result: Success
7
8  -- INVALID INSERT: Double room price <= highest single (80)

```

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

## 6.4 Constraint (c): No Overlapping Bookings

```
1 DROP TRIGGER IF EXISTS trg_no_overlapping_bookings_insert;
2 DELIMITER //
3 CREATE TRIGGER trg_no_overlapping_bookings_insert
4 BEFORE INSERT ON Booking
5 FOR EACH ROW
6 BEGIN
7     DECLARE overlap_count INT;
8
9     SELECT COUNT(*) INTO overlap_count
10    FROM Booking
11   WHERE guestNo = NEW.guestNo
12         AND NOT (NEW.dateTo <= dateFrom OR NEW.dateFrom >= dateTo);
13
14     IF overlap_count > 0 THEN
15         SIGNAL SQLSTATE '45000'
16         SET MESSAGE_TEXT = 'Error: Guest cannot have overlapping
17         bookings.';
18     END IF;
19 END //
20 DELIMITER ;
```

### Test Validation

```
1 -- Setup test data
2 INSERT INTO Guest VALUES (1, 'John Doe', '123 Main St', 0);
3
4 -- First booking: Jan 1-5
5 INSERT INTO Booking VALUES (1, '2024-01-01', 101, 1, '2024-01-05', 2);
6 -- Result: Success
7
8 -- VALID INSERT: Non-overlapping (Jan 10-15)
9 INSERT INTO Booking VALUES (1, '2024-01-10', 102, 1, '2024-01-15', 1);
10 -- Result: Success
11
12 -- INVALID INSERT: Overlapping with first booking (Jan 3-8)
13 INSERT INTO Booking VALUES (1, '2024-01-03', 103, 1, '2024-01-08', 1);
14 -- Result: Error - Guest cannot have overlapping bookings.
```

## 6.5 Constraint (d): NumOfAdult ≤ NumAdultMax

```
1 DROP TRIGGER IF EXISTS trg_check_num_adults_insert;
2 DELIMITER //
3 CREATE TRIGGER trg_check_num_adults_insert
4 BEFORE INSERT ON Booking
5 FOR EACH ROW
6 BEGIN
7     DECLARE max_adults INT;
8
```

```

9      SELECT NumAdultMax INTO max_adults
10     FROM Room WHERE roomNo = NEW.roomNo AND hotelNo = NEW.hotelNo;
11
12     IF NEW.NumOfAdult > max_adults THEN
13         SIGNAL SQLSTATE '45000'
14         SET MESSAGE_TEXT = 'Error: Number of adults exceeds room
15         capacity.';
16     END IF;
17 END //
DELIMITER ;

```

### Test Validation

```

1 -- Assume Room 101 has NumAdultMax = 2
2 -- VALID INSERT: 2 adults in room with max 2
3 INSERT INTO Booking VALUES (1, '2024-02-01', 101, 1, '2024-02-05', 2);
4 -- Result: Success
5
6 -- INVALID INSERT: 4 adults in room with max 2
7 INSERT INTO Booking VALUES (1, '2024-03-01', 101, 1, '2024-03-05', 4);
8 -- Result: Error - Number of adults exceeds room capacity.

```

## 6.6 Constraint (e): Auto-Calculate TotalAmount

```

1 DROP TRIGGER IF EXISTS trg_update_total_amount_insert;
2 DELIMITER //
3 CREATE TRIGGER trg_update_total_amount_insert
4 AFTER INSERT ON Booking
5 FOR EACH ROW
6 BEGIN
7     DECLARE room_price DECIMAL(10, 2);
8     DECLARE num_days INT;
9     DECLARE booking_cost DECIMAL(12, 2);
10
11     SELECT price INTO room_price
12     FROM Room WHERE roomNo = NEW.roomNo AND hotelNo = NEW.hotelNo;
13
14     SET num_days = DATEDIFF(NEW.dateTo, NEW.dateFrom);
15     SET booking_cost = room_price * num_days;
16
17     UPDATE Guest
18     SET TotalAmount = TotalAmount + booking_cost
19     WHERE guestNo = NEW.guestNo;
20 END //
21 DELIMITER ;

```

### Test Validation

```

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

```

Output Before:

guestNo	guestName	TotalAmount
1	John Doe	0.00

```

1 -- Insert booking: Room price = 150, 5 days = 750
2 INSERT INTO Booking VALUES (1, '2024-04-01', 101, 1, '2024-04-06', 2);
3
4 -- Check guest's TotalAmount after booking
5 SELECT guestNo, guestName, TotalAmount FROM Guest WHERE guestNo = 1;

```

Output After:

guestNo	guestName	TotalAmount
1	John Doe	750.00

## 6.7 Constraint (f): INSTEAD OF Trigger for LondonHotelRoom View

```

1 -- Create the view
2 DROP VIEW IF EXISTS LondonHotelRoom;
3 CREATE VIEW LondonHotelRoom AS
4 SELECT h.hotelNo, hotelName, city, roomNo, type, price
5 FROM Hotel h, Room r
6 WHERE h.hotelNo = r.hotelNo AND city = 'London';
7
8 -- MySQL doesn't support INSTEAD OF triggers, use stored procedure
9 DROP PROCEDURE IF EXISTS InsertIntoLondonHotelRoom;
10 DELIMITER //
11 CREATE PROCEDURE InsertIntoLondonHotelRoom(
12     IN p_hotelNo INT,
13     IN p_hotelName VARCHAR(50),
14     IN p_roomNo INT,
15     IN p_type VARCHAR(20),
16     IN p_price DECIMAL(10, 2)
17 )
18 BEGIN
19     DECLARE hotel_exists INT;
20
21     SELECT COUNT(*) INTO hotel_exists FROM Hotel WHERE hotelNo =
22         p_hotelNo;
23
24     IF hotel_exists = 0 THEN
25         INSERT INTO Hotel (hotelNo, hotelName, city)
26         VALUES (p_hotelNo, p_hotelName, 'London');
27     ELSE
28         UPDATE Hotel SET hotelName = p_hotelName WHERE hotelNo =
29             p_hotelNo;
30     END IF;
31
32     INSERT INTO Room (roomNo, hotelNo, type, price)
33     VALUES (p_roomNo, p_hotelNo, p_type, p_price);
34 END //
35 DELIMITER ;

```

### Test Validation

```

1 -- Query the view
2 SELECT * FROM LondonHotelRoom;

```

Expected Output:



hotelNo	hotelName	city	roomNo	type	price
1	Test Hotel	London	101	double	150.00
1	Test Hotel	London	103	single	50.00

```

1 -- Use procedure to insert via the view
2 CALL InsertIntoLondonHotelRoom(2, 'Grand Hotel', 301, 'double', 200.00)
3 ;
4 -- Verify the insert
5 SELECT * FROM LondonHotelRoom WHERE hotelNo = 2;

```

**Expected Output:**

hotelNo	hotelName	city	roomNo	type	price
2	Grand Hotel	London	301	double	200.00

## 7 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.