Exercise 1

Create a database schema for a university Database.

AIM :

The aim of this database schema is to store and manage information about students, professors, courses, and enrollments at a university.

Procedures :

Step 1: Create Database

```
CREATE  DATABASE  SRMUniversity;
```

Step 2: Use Database.

```
USE SRMUniversity;
```

Step 3: Create Tables of Departments, Students, Faculty, Courses, Classes, Enrollment, Attendance, Grades.

```
CREATE TABLE Departments (department_id INT PRIMARY KEY,department_name VARCHAR(100) NOT NULL,building VARCHAR(100));

CREATE TABLE Students (student_id INT PRIMARY KEY, Name VARCHAR(50) NOT NULL, dob DATE NOT NULL, phone_number VARCHAR(15), department_id INT, FOREIGN KEY (department_id) REFERENCES Departments(department_id));

CREATE TABLE Faculty (faculty_id INT PRIMARY KEY,Name VARCHAR(50) NOT NULL, phone_number VARCHAR(15),position VARCHAR(50),department_id INT,FOREIGN KEY (department_id) REFERENCES Departments(department_id));

CREATE TABLE Courses (course_id INT PRIMARY KEY,course_name VARCHAR(100) NOT NULL,credit_hours INT NOT NULL,department_id INT,FOREIGN KEY (department_id) REFERENCES Departments(department_id));

CREATE TABLE Classes (class_id INT PRIMARY KEY,course_id INT NOT NULL,faculty_id INT NOT NULL,schedule VARCHAR(100),room_number VARCHAR(10),semester VARCHAR(10),year YEAR,FOREIGN KEY
(course_id) REFERENCES Courses(course_id),FOREIGN KEY (faculty_id) REFERENCES Faculty(faculty_id));

CREATE TABLE Enrollment (enrollment_id INT PRIMARY KEY,student_id INT NOT NULL,class_id INT NOT NULL,enrollment_date DATE,FOREIGN KEY (student_id) REFERENCES Students(student_id),FOREIGN KEY (class_id) REFERENCES Classes(class_id));
```

CREATE TABLE Attendance (attendance_id INT PRIMARY KEY,student_id INT NOT NULL,class_id INT NOT NULL,attendance_date DATE NOT NULL,status ENUM('Present', 'Absent') NOT NULL,FOREIGN KEY (student_id) REFERENCES Students(student_id),FOREIGN KEY (class_id) REFERENCES Classes(class_id));

CREATE TABLE Grades (grade_id INT PRIMARY KEY,student_id INT NOT NULL,class_id INT NOT NULL,grade CHAR(1) CHECK (grade IN ('A', 'B', 'C', 'D', 'F')),FOREIGN KEY (student_id) REFERENCES Students(student_id),FOREIGN KEY (class_id) REFERENCES Classes(class_id));

Step 4: Inserting Values into Tables.

INSERT INTO Departments (department_id, department_name, building)VALUES (1,'Computer Science', 'Engineering Block A'),(2,'Mathematics', 'Science Block B'),(3,'Physics', 'Science Block C');

INSERT INTO Students (student_id,Name, dob, phone_number,department_id)VALUES (1,'abcd', '2000-01-15', '1234567890', 1),(2,'efghr', '2001-08-10', '1122334455', 2),(3,'ijkl', '1999-05-20', '16534848', 3);

INSERT INTO Faculty (faculty_id,Name,phone_number, position, department_id)VALUES (1,'Dr. xyz', '1234556', 'Professor', 1),(2,'xzy','8765432109', 'Lecturer', 2),(3,'wxy', '7654321098', 'Professor', 3);

INSERT INTO Courses (course_id,course_name, credit_hours, department_id)VALUES (102,'Data Structures', 3, 1),(201,'Linear Algebra', 4, 2),(211,'Quantum Mechanics', 3, 3);

INSERT INTO Classes (class_id,course_id, faculty_id, schedule, room_number, semester, year)VALUES (1,102, 1, 'Monday 10:00 AM - 12:00 PM', 'Room 101', 'Fall', 2024),(2,201, 2, 'Tuesday 2:00 PM - 4:00 PM', 'Room 202', 'Fall', 2024),(3,211, 3, 'Wednesday 9:00 AM - 11:00 AM', 'Room 303', 'Fall', 2024);

INSERT INTO Enrollment (enrollment_id,student_id, class_id, enrollment_date)VALUES (1,1, 1, '2024-08-15'),(2,2, 2, '2024-08-16'),(3,3, 3, '2024-08-17');

INSERT INTO Attendance (attendance_id,student_id, class_id, attendance_date, status)VALUES (1,1, 1, '2024-09-01', 'Present'),(2,2, 2, '2024-09-01', 'Absent'),(3,3, 3, '2024-09-01', 'Present');

INSERT INTO Grades (grade_id,student_id, class_id, grade)VALUES (1,1, 1, 'A'),(2,2, 2, 'B'),(3,3, 3, 'A');

Step 5: Display Tables. SELECT *

FROM Departments; SELECT *

FROM Students; SELECT * FROM

Faculty; SELECT * FROM Courses;

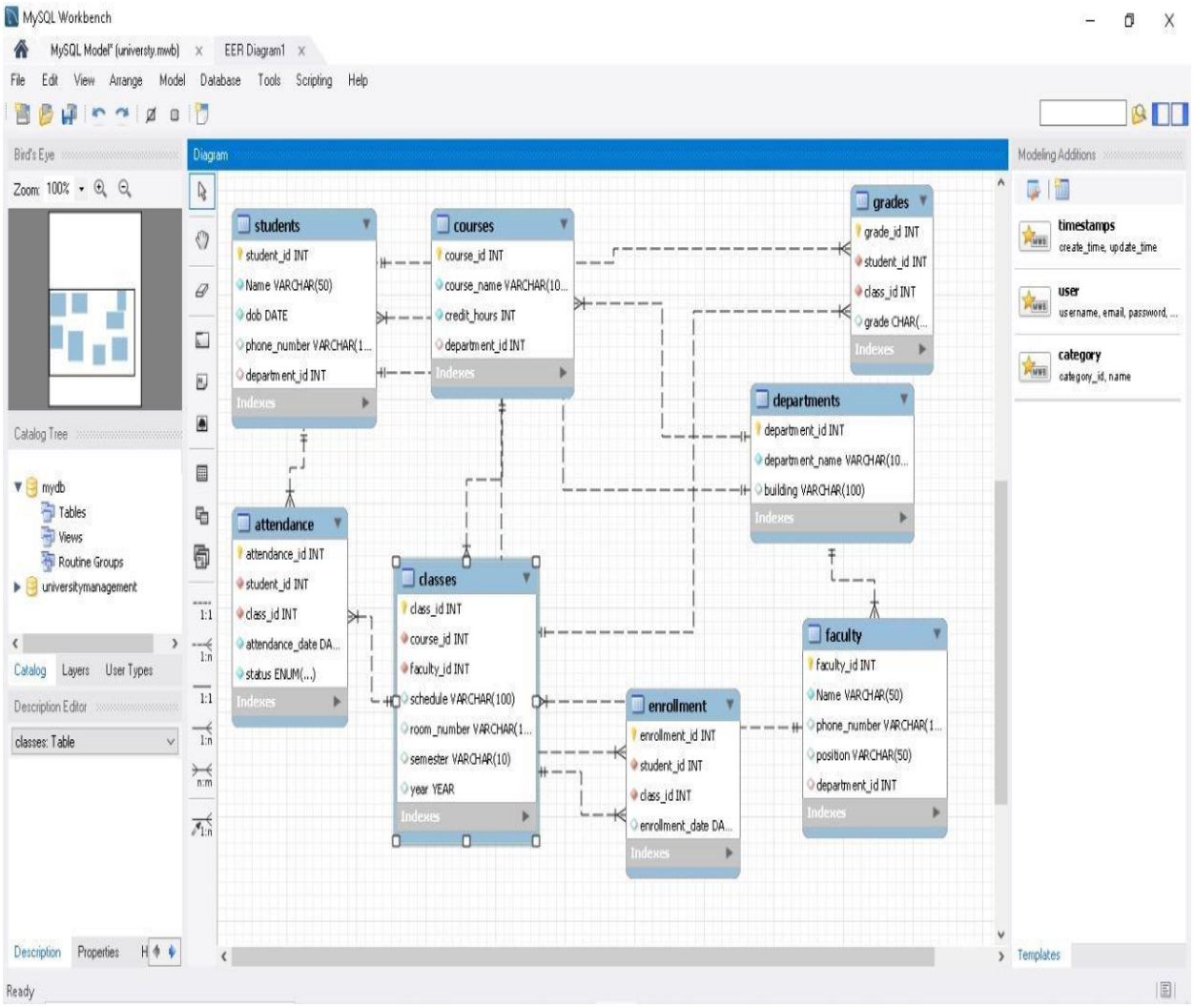
SELECT * FROM Classes; SELECT

* FROM Enrollment; SELECT *

FROM Attendance; SELECT *

FROM Grades;

OUTPUT



RESULT:

Thus the queries are executed successfully and university database schema is created.

Exercise 2

Create employee database with key constraints

AIM :

The aim of this database schema is to store and manage information about employees with key constraints using SQL queries.

Procedures :

Step 1: Create Database

```
Create database empdb;
```

Step 2: Use Database.

```
use empdb;
```

Step 3: Create Tables of Departments and Employees

```
CREATE TABLE Departments (
    DepartmentID INT PRIMARY KEY,
    DepartmentName VARCHAR(100) NOT NULL UNIQUE
);
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    Email VARCHAR(100) NOT NULL UNIQUE,
    Phone VARCHAR(15),
    HireDate DATE NOT NULL,
    Salary DECIMAL(10, 2),
    DepartmentID INT,
    CONSTRAINT fk_Department FOREIGN KEY (DepartmentID) REFERENCES
Departments(DepartmentID)
```

);

STEP 4: Insert values into Department and Employees

```
INSERT INTO Departments (DepartmentID, DepartmentName)

VALUES

    (1, 'HR'),

    (2, 'Finance'),

    (3, 'IT');


INSERT INTO Employees (EmployeeID, FirstName, LastName, Email, Phone, HireDate, Salary, DepartmentID)

VALUES

    (101, 'Alice', 'Johnson', 'alice.johnson@example.com', '555-1234', '2023-01-15', 60000, 1),

    (102, 'Bob', 'Smith', 'bob.smith@example.com', '555-5678', '2022-06-01', 75000, 3),

    (103, 'Charlie', 'Brown', 'charlie.brown@example.com', '555-8765', '2024-03-20', 50000, 2);
```

STEP 5: Display Tables

```
SELECT * FROM Employees;

SELECT * FROM Departments ;
```

STEP 6 : Alter Tables

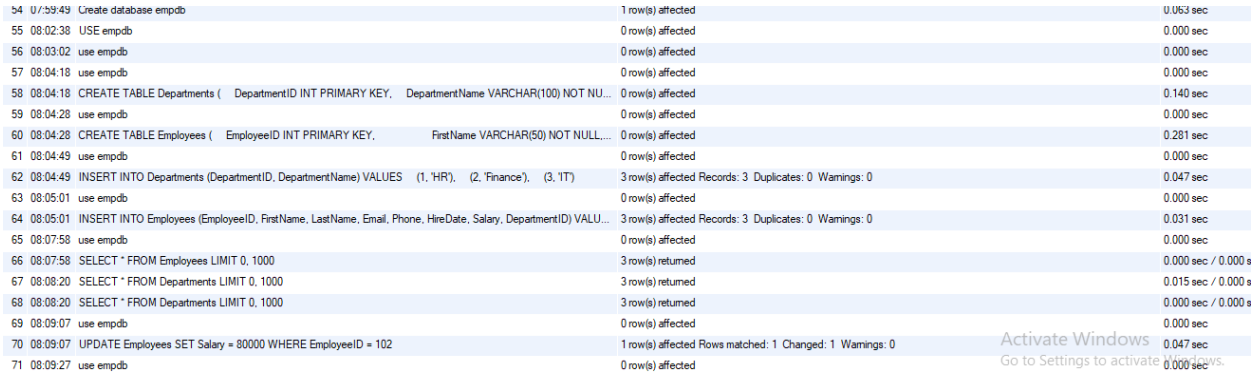
```
UPDATE Employees

SET Salary = 80000

WHERE EmployeeID = 102;
```

UPDATE Employees

SET DepartmentID = 3

WHERE EmployeeID = 103;

OUTPUT



54	07:59:49	Create database empdb	1 row(s) affected
55	08:02:38	USE empdb	0 row(s) affected
56	08:03:02	use empdb	0 row(s) affected
57	08:04:18	use empdb	0 row(s) affected
58	08:04:18	CREATE TABLE Departments (DepartmentID INT PRIMARY KEY, DepartmentName VARCHAR(100) NOT NU...	0 row(s) affected
59	08:04:28	use empdb	0 row(s) affected
60	08:04:28	CREATE TABLE Employees (EmployeeID INT PRIMARY KEY, FirstName VARCHAR(50) NOT NULL,...	0 row(s) affected
61	08:04:49	use empdb	0 row(s) affected
62	08:04:49	INSERT INTO Departments (DepartmentID, DepartmentName) VALUES (1, 'HR'), (2, 'Finance'), (3, 'IT')	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0
63	08:05:01	use empdb	0 row(s) affected
64	08:05:01	INSERT INTO Employees (EmployeeID, FirstName, LastName, Email, Phone, HireDate, Salary, DepartmentID) VALU...	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0
65	08:07:58	use empdb	0 row(s) affected
66	08:07:58	SELECT * FROM Employees LIMIT 0, 1000	3 row(s) returned
67	08:08:20	SELECT * FROM Departments LIMIT 0, 1000	3 row(s) returned
68	08:08:20	SELECT * FROM Departments LIMIT 0, 1000	3 row(s) returned
69	08:09:07	use empdb	0 row(s) affected
70	08:09:07	UPDATE Employees SET Salary = 80000 WHERE EmployeeID = 102	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0
71	08:09:27	use empdb	0 row(s) affected

	DepartmentID	DepartmentName
▶	2	Finance
	1	HR
	3	IT
*	NULL	NULL

Result Grid | Filter Rows: | Edit: | Export/Import:

	EmployeeID	FirstName	LastName	Email	Phone	HireDate
▶	101	Alice	Johnson	alice.johnson@example.com	555-1234	2023-01-15
	102	Bob	Smith	bob.smith@example.com	555-5678	2022-06-01
	103	Charlie	Brown	charlie.brown@example.com	555-8765	2024-03-20
*	NULL	NULL	NULL	NULL	NULL	NULL

RESULT:

Thus the queries are executed successfully and employee database with key constraints was created.

Exercise 3

Create ER Model University Database

AIM :

The aim is to create University database ER model using MYSQL workbench.

Procedures :

Step 1. Open MySQL Workbench

- Launch MySQL Workbench.
- Create a new model by selecting **File > New Model**.
- Click **Add Diagram** to create a new EER Diagram.

Step 2. Add Tables for Each Entity

- From the vertical toolbar, drag the **Table** icon into the canvas for each entity:
    - Student
    - Course
    - Department
    - Instructor
    - Enrollment
    - Classroom
    - Schedule

Identify Entities and Their Attributes

Entity	Attributes
Student	StudentID (PK), FirstName, LastName, DateOfBirth, Email (Unique), EnrollmentDate
Course	CourseID (PK), CourseName, Credits, DepartmentID (FK)
Department	DepartmentID (PK), DepartmentName
Instructor	InstructorID (PK), FirstName, LastName, Email (Unique), DepartmentID (FK)

Entity	Attributes
Enrollment	EnrollmentID (PK), StudentID (FK), CourseID (FK), EnrollmentDate, Grade
Classroom	ClassroomID (PK), Building, RoomNumber, Capacity
Schedule	ScheduleID (PK), CourseID (FK), InstructorID (FK), ClassroomID (FK), DayOfWeek, StartTime, EndTime

Step 3. Define Columns and Keys for Each Table

- Double-click each table to open the table editor.
- Add columns according to the attributes planned.
- For each table:
    - Set the primary key (PK) by checking the **PK** checkbox next to the relevant attribute (e.g., StudentID).
    - Set unique constraints for unique fields (e.g., Email in Student and Instructor).

Step 4. Create Foreign Key Relationships

- Still in the table editor, go to the **Foreign Keys** tab.
- Create foreign keys by linking attributes to the referenced table's primary key.
    - Example: In `Course`, set `DepartmentID` as a foreign key referencing `Department.DepartmentID`.
    - Similarly set foreign keys for Enrollment (StudentID, CourseID), Instructor (DepartmentID), Schedule (CourseID, InstructorID, ClassroomID).

Step 5. Use Relationship Tools to Connect Tables Visually

- Use the **One-to-Many** or **Many-to-Many** relationship tools from the toolbar to visually link the tables.
- For many-to-many relationships (like Student-Course), ensure you use the associative entity **Enrollment**.

DEFINE relationship

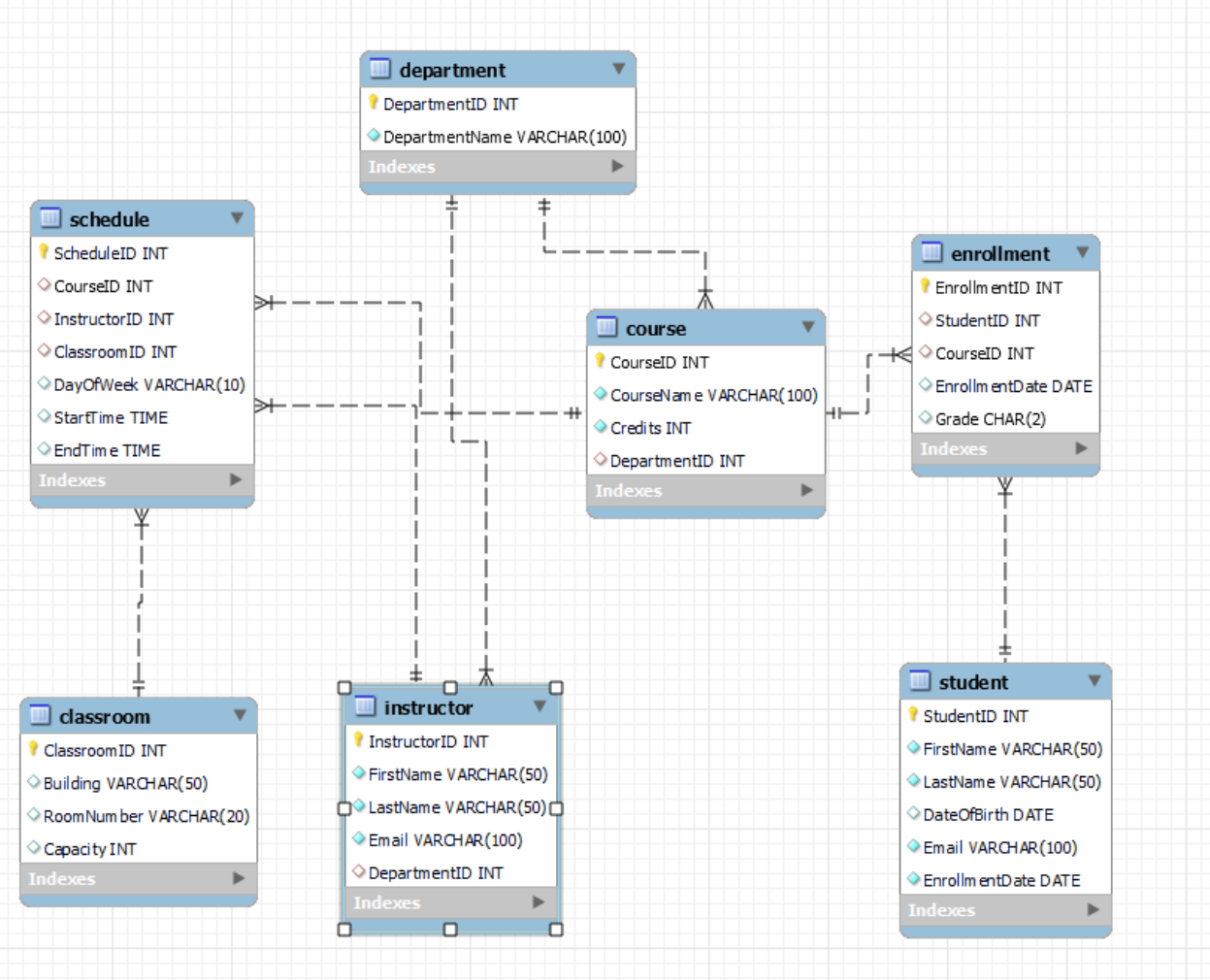
- **Department** has many **Courses** and many **Instructors**.
- **Student** enrolls in many **Courses** (via **Enrollment**).
- **Course** is taught by many **Instructors** (via **Schedule**).
- **Course** scheduled in a **Classroom**.

- **Instructor** belongs to one **Department**.

Step 6. Save and Export

- Save your model regularly (`File > Save`).
- Export your diagram as an image or PDF using `File > Export > Export as PNG/PDF` if you want to share or print the ER diagram.

OUTPUT



RESULT:

Thus ER model for University database is created successfully using MYSQL workbench.

Exercise 4

Implementing DDL and DML Commands

AIM :

To understand and implement Data Definition Language (DDL) and Data Manipulation Language (DML) commands using SQL.

Procedures :

DDL (Data Definition Language)

Commands used to define and modify database structure.
Examples:

- CREATE
- ALTER
- DROP
- TRUNCATE

DML (Data Manipulation Language):

Commands used to manipulate data in tables.
Examples:

- INSERT
- UPDATE
- DELETE

A. DDL Commands (Data Definition Language)

Step 1: Create a Table

Create a table named `Students` with fields for student details.

```
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(50),
    Age INT,
    Department VARCHAR(30)
);
```

Step 2: Alter the Table

Add a new column `Email` to the `Students` table.

```
ALTER TABLE Students
```

```
ADD Email VARCHAR(50);
```

Step 3: Truncate the Table

Remove all records from the `Students` table, keeping its structure.

```
TRUNCATE TABLE Students;
```

Step 4: Drop the Table

Permanently delete the `Students` table from the database.

```
DROP TABLE Students;
```

B. DML Commands (Data Manipulation Language)

Step 5: Insert Data into the Table

Insert a record for a student into the `Students` table.

```
INSERT INTO Students (StudentID, Name, Age, Department, Email)
VALUES (1, 'John Doe', 20, 'CSE', 'john@example.com');
```

Step 6: Retrieve Data from the Table

Display all records from the `Students` table.

```
SELECT * FROM Students;
```

Step 7: Update a Record

Update the email address of the student with `StudentID = 1`.

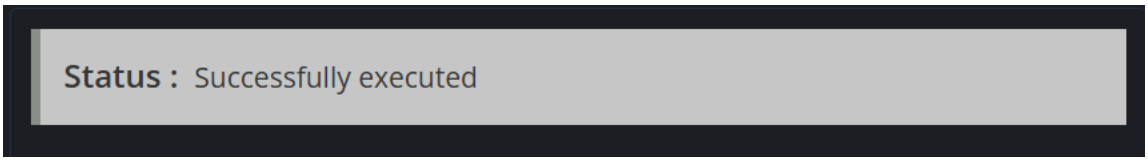
```
UPDATE Students
SET Email = 'john.doe@college.com'
WHERE StudentID = 1;
```

Step 8: Delete a Record

Delete the record of the student with `StudentID = 1`.

```
DELETE FROM Students
WHERE StudentID = 1;
```

OUTPUT:



RESULT:

Thus the SQL commands for managing database schema (DDL) and manipulating records (DML) are executed successfully.

Exercise 5

Implement DCL,TCL Commands

AIM :

To understand and implement DCL and TCL commands using SQL for managing access permissions and transactions using SQL.

Procedures :

DCL – Data Control Language

Used to control access to data in a database.
Common commands:

- `GRANT` – gives privileges to users
- `REVOKE` – removes privileges from users

TCL – Transaction Control Language

Used to manage transactions in a database.
Common commands:

- `COMMIT` – saves the transaction permanently
- `ROLLBACK` – undoes changes since the last COMMIT
- `SAVEPOINT` – creates a point within a transaction for partial rollbacks

A. DCL Commands

Step 1: Create a User (syntax may vary depending on RDBMS)

```
CREATE USER 'student1'@'localhost' IDENTIFIED BY 'password123';
```

Step 2: Grant Privileges

Grant permissions to the user to access a specific table.

```
GRANT SELECT, INSERT, UPDATE ON empdb.Employees TO 'student1'@'localhost';
```

Step 3: Revoke Privileges

Revoke the granted permissions.

```
REVOKE INSERT, UPDATE ON empdb.Employees FROM 'student1'@'localhost';
```

B. TCL Commands

Step 4: Start a Transaction (optional depending on system)

```
START TRANSACTION;
```

Step 5: Insert Data into a Table

Insert a new employee (example assumes `Employees` table exists).

```
INSERT INTO Employees (EmployeeID, FirstName, LastName, Email, Phone,
HireDate, Salary, DepartmentID)
VALUES (104, 'David', 'Lee', 'david.lee@example.com', '555-1111', '2025-08-
03', 62000, 2);
```

Step 6: Create a Savepoint

Create a rollback point.

```
SAVEPOINT sp1;
```

Step 7: Update a Record

Update the inserted employee's salary.

```
UPDATE Employees
SET Salary = 65000
WHERE EmployeeID = 104;
```

Step 8: Rollback to Savepoint

Undo the salary update, but keep the insert.

```
ROLLBACK TO sp1;
```

Step 9: Commit the Transaction

Make all changes permanent.

```
COMMIT;
```

Step 10: Full Rollback Example

If no `COMMIT` is done, you can rollback all changes:

```
ROLLBACK;
```

OUTPUT:

RESULT:

Thus, DCL and TCL commands were executed successfully for managing database security and transaction control.

Exercise 6

Implement SQL sub queries, Joins and Clauses

AIM :

To implement SQL subqueries, various types of joins, and commonly used clauses for effective data retrieval and management.

Procedures :

Subqueries:

A subquery is a query nested inside another query.
Types:

- Single-row subquery
- Multi-row subquery
- Correlated subquery

Joins:

Used to retrieve data from multiple tables based on a related column.
Types:

- `INNER JOIN`
- `LEFT JOIN` (or LEFT OUTER JOIN)
- `RIGHT JOIN` (or RIGHT OUTER JOIN)
- `FULL JOIN` *(may not be supported in all DBMSs)*
- `SELF JOIN`

Clauses:

SQL clauses filter, sort, and group query results. Common clauses include:

- WHERE
- GROUP BY
- HAVING
- ORDER BY
- LIMIT

Tables for Practice:

Departments

```
CREATE TABLE Departments (
    DepartmentID INT PRIMARY KEY,
    DepartmentName VARCHAR(100)
);
```

Employees

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Salary DECIMAL(10, 2),
    DepartmentID INT,
    FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)
);
```

INSERT INTO Departments

```
INSERT INTO Departments (DepartmentID, DepartmentName)
VALUES
    (1, 'HR'),
    (2, 'Finance'),
    (3, 'IT'),
    (4, 'Marketing');
```

INSERT INTO Employees

```
INSERT INTO Employees (EmployeeID, FirstName, LastName, Salary, DepartmentID)
VALUES
    (101, 'Alice', 'Johnson', 55000.00, 1),
    (102, 'Bob', 'Smith', 72000.00, 3),
    (103, 'Charlie', 'Brown', 48000.00, 2),
    (104, 'David', 'Lee', 65000.00, 3),
    (105, 'Eva', 'Green', 51000.00, 4),
    (106, 'Frank', 'Miller', 75000.00, 2);
```

A. Subqueries

Step 1: Subquery in `WHERE` clause

Get employees earning more than the average salary.

```
SELECT * FROM Employees
WHERE Salary > (SELECT AVG(Salary) FROM Employees);
```

Step 2: Subquery in `FROM` clause

Get department-wise average salary using a subquery.

```
SELECT DepartmentID, AvgSalary
FROM (SELECT DepartmentID, AVG(Salary) AS AvgSalary FROM Employees GROUP BY
DepartmentID) AS DeptAvg;
```

B. Joins

Step 3: INNER JOIN

Get employee details with their department names.

```
SELECT e.EmployeeID, e.FirstName, e.LastName, d.DepartmentName
FROM Employees e
INNER JOIN Departments d ON e.DepartmentID = d.DepartmentID;
```

Step 4: LEFT JOIN

Show all employees and their departments, even if the department is NULL.

```
SELECT e.EmployeeID, e.FirstName, d.DepartmentName
FROM Employees e
LEFT JOIN Departments d ON e.DepartmentID = d.DepartmentID;
```

Step 5: RIGHT JOIN

Show all departments and employees, even if no employees are in a department.

```
SELECT e.EmployeeID, e.FirstName, d.DepartmentName
FROM Employees e
RIGHT JOIN Departments d ON e.DepartmentID = d.DepartmentID;
```

Step 6: SELF JOIN

Find pairs of employees from the same department.

```
SELECT e1.FirstName AS Emp1, e2.FirstName AS Emp2, e1.DepartmentID
FROM Employees e1
JOIN Employees e2 ON e1.DepartmentID = e2.DepartmentID AND e1.EmployeeID <>
e2.EmployeeID;
```

C. SQL Clauses

Step 7: WHERE Clause

Get employees from the IT department.

```
SELECT * FROM Employees
WHERE DepartmentID = 3;
```

Step 8: GROUP BY Clause

Find total salary paid in each department.

```
SELECT DepartmentID, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY DepartmentID;
```

Step 9: HAVING Clause

Departments where total salary exceeds 1,00,000.

```
SELECT DepartmentID, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY DepartmentID
HAVING SUM(Salary) > 100000;
```

Step 10: ORDER BY Clause

Display all employees in descending order of salary.

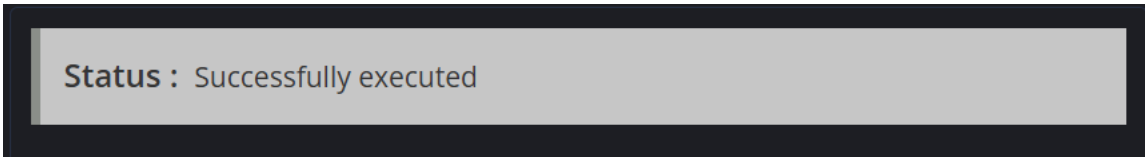
```
SELECT * FROM Employees
ORDER BY Salary DESC;
```

Step 11: LIMIT Clause (MySQL)

Fetch top 3 highest paid employees.

```
SELECT * FROM Employees
ORDER BY Salary DESC
LIMIT 3;
```

OUTPUT:



RESULT:

Thus, SQL subqueries, joins, and clauses were implemented successfully using structured SQL queries.

Exercise 4

Implementing DDL and DML Commands

AIM :

To understand and implement Data Definition Language (DDL) and Data Manipulation Language (DML) commands using SQL.

Procedures :

DDL (Data Definition Language)

Commands used to define and modify database structure.
Examples:

- CREATE
- ALTER
- DROP
- TRUNCATE

DML (Data Manipulation Language):

Commands used to manipulate data in tables.
Examples:

- INSERT
- UPDATE
- DELETE

A. DDL Commands (Data Definition Language)

Step 1: Create a Table

Create a table named `Students` with fields for student details.

```
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(50),
    Age INT,
    Department VARCHAR(30)
);
```

Step 2: Alter the Table

Add a new column `Email` to the `Students` table.

```
ALTER TABLE Students
```

```
ADD Email VARCHAR(50);
```

Step 3: Truncate the Table

Remove all records from the `Students` table, keeping its structure.

```
TRUNCATE TABLE Students;
```

Step 4: Drop the Table

Permanently delete the `Students` table from the database.

```
DROP TABLE Students;
```

B. DML Commands (Data Manipulation Language)

Step 5: Insert Data into the Table

Insert a record for a student into the `Students` table.

```
INSERT INTO Students (StudentID, Name, Age, Department, Email)
VALUES (1, 'John Doe', 20, 'CSE', 'john@example.com');
```

Step 6: Retrieve Data from the Table

Display all records from the `Students` table.

```
SELECT * FROM Students;
```

Step 7: Update a Record

Update the email address of the student with `StudentID = 1`.

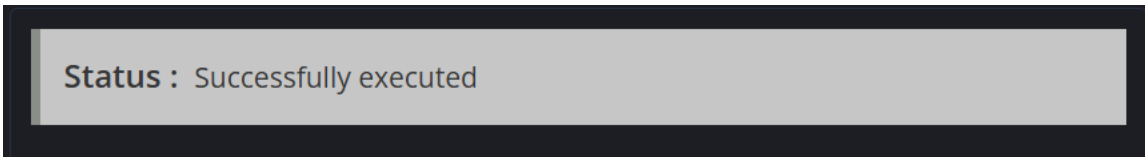
```
UPDATE Students
SET Email = 'john.doe@college.com'
WHERE StudentID = 1;
```

Step 8: Delete a Record

Delete the record of the student with `StudentID = 1`.

```
DELETE FROM Students
WHERE StudentID = 1;
```

OUTPUT:



RESULT:

Thus the SQL commands for managing database schema (DDL) and manipulating records (DML) are executed successfully.

Exercise 5

Implement DCL,TCL Commands

AIM :

To understand and implement DCL and TCL commands using SQL for managing access permissions and transactions using SQL.

Procedures :

DCL – Data Control Language

Used to control access to data in a database.
Common commands:

- `GRANT` – gives privileges to users
- `REVOKE` – removes privileges from users

TCL – Transaction Control Language

Used to manage transactions in a database.
Common commands:

- `COMMIT` – saves the transaction permanently
- `ROLLBACK` – undoes changes since the last COMMIT
- `SAVEPOINT` – creates a point within a transaction for partial rollbacks

A. DCL Commands

Step 1: Create a User (syntax may vary depending on RDBMS)

```
CREATE USER 'student1'@'localhost' IDENTIFIED BY 'password123';
```

Step 2: Grant Privileges

Grant permissions to the user to access a specific table.

```
GRANT SELECT, INSERT, UPDATE ON empdb.Employees TO 'student1'@'localhost';
```

Step 3: Revoke Privileges

Revoke the granted permissions.

```
REVOKE INSERT, UPDATE ON empdb.Employees FROM 'student1'@'localhost';
```

B. TCL Commands

Step 4: Start a Transaction (optional depending on system)

```
START TRANSACTION;
```

Step 5: Insert Data into a Table

Insert a new employee (example assumes `Employees` table exists).

```
INSERT INTO Employees (EmployeeID, FirstName, LastName, Email, Phone,
HireDate, Salary, DepartmentID)
VALUES (104, 'David', 'Lee', 'david.lee@example.com', '555-1111', '2025-08-
03', 62000, 2);
```

Step 6: Create a Savepoint

Create a rollback point.

```
SAVEPOINT sp1;
```

Step 7: Update a Record

Update the inserted employee's salary.

```
UPDATE Employees
SET Salary = 65000
WHERE EmployeeID = 104;
```

Step 8: Rollback to Savepoint

Undo the salary update, but keep the insert.

```
ROLLBACK TO sp1;
```

Step 9: Commit the Transaction

Make all changes permanent.

```
COMMIT;
```

Step 10: Full Rollback Example

If no `COMMIT` is done, you can rollback all changes:

```
ROLLBACK;
```

OUTPUT:

RESULT:

Thus, DCL and TCL commands were executed successfully for managing database security and transaction control.

Exercise 6

Implement SQL sub queries, Joins and Clauses

AIM :

To implement SQL subqueries, various types of joins, and commonly used clauses for effective data retrieval and management.

Procedures :

Subqueries:

A subquery is a query nested inside another query.
Types:

- Single-row subquery
- Multi-row subquery
- Correlated subquery

Joins:

Used to retrieve data from multiple tables based on a related column.
Types:

- `INNER JOIN`
- `LEFT JOIN` (or LEFT OUTER JOIN)
- `RIGHT JOIN` (or RIGHT OUTER JOIN)
- `FULL JOIN` *(may not be supported in all DBMSs)*
- `SELF JOIN`

Clauses:

SQL clauses filter, sort, and group query results. Common clauses include:

- WHERE
- GROUP BY
- HAVING
- ORDER BY
- LIMIT

Tables for Practice:

Departments

```
CREATE TABLE Departments (
    DepartmentID INT PRIMARY KEY,
    DepartmentName VARCHAR(100)
);
```

Employees

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Salary DECIMAL(10, 2),
    DepartmentID INT,
    FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)
);
```

INSERT INTO Departments

```
INSERT INTO Departments (DepartmentID, DepartmentName)
VALUES
    (1, 'HR'),
    (2, 'Finance'),
    (3, 'IT'),
    (4, 'Marketing');
```

INSERT INTO Employees

```
INSERT INTO Employees (EmployeeID, FirstName, LastName, Salary, DepartmentID)
VALUES
    (101, 'Alice', 'Johnson', 55000.00, 1),
    (102, 'Bob', 'Smith', 72000.00, 3),
    (103, 'Charlie', 'Brown', 48000.00, 2),
    (104, 'David', 'Lee', 65000.00, 3),
    (105, 'Eva', 'Green', 51000.00, 4),
    (106, 'Frank', 'Miller', 75000.00, 2);
```

A. Subqueries

Step 1: Subquery in `WHERE` clause

Get employees earning more than the average salary.

```
SELECT * FROM Employees
WHERE Salary > (SELECT AVG(Salary) FROM Employees);
```

Step 2: Subquery in `FROM` clause

Get department-wise average salary using a subquery.

```
SELECT DepartmentID, AvgSalary
FROM (SELECT DepartmentID, AVG(Salary) AS AvgSalary FROM Employees GROUP BY
DepartmentID) AS DeptAvg;
```

B. Joins

Step 3: INNER JOIN

Get employee details with their department names.

```
SELECT e.EmployeeID, e.FirstName, e.LastName, d.DepartmentName
FROM Employees e
INNER JOIN Departments d ON e.DepartmentID = d.DepartmentID;
```

Step 4: LEFT JOIN

Show all employees and their departments, even if the department is NULL.

```
SELECT e.EmployeeID, e.FirstName, d.DepartmentName
FROM Employees e
LEFT JOIN Departments d ON e.DepartmentID = d.DepartmentID;
```

Step 5: RIGHT JOIN

Show all departments and employees, even if no employees are in a department.

```
SELECT e.EmployeeID, e.FirstName, d.DepartmentName
FROM Employees e
RIGHT JOIN Departments d ON e.DepartmentID = d.DepartmentID;
```

Step 6: SELF JOIN

Find pairs of employees from the same department.

```
SELECT e1.FirstName AS Emp1, e2.FirstName AS Emp2, e1.DepartmentID
FROM Employees e1
JOIN Employees e2 ON e1.DepartmentID = e2.DepartmentID AND e1.EmployeeID <>
e2.EmployeeID;
```

C. SQL Clauses

Step 7: WHERE Clause

Get employees from the IT department.

```
SELECT * FROM Employees
WHERE DepartmentID = 3;
```

Step 8: GROUP BY Clause

Find total salary paid in each department.

```
SELECT DepartmentID, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY DepartmentID;
```

Step 9: HAVING Clause

Departments where total salary exceeds 1,00,000.

```
SELECT DepartmentID, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY DepartmentID
HAVING SUM(Salary) > 100000;
```

Step 10: ORDER BY Clause

Display all employees in descending order of salary.

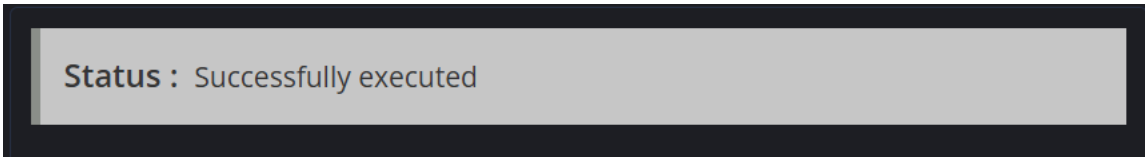
```
SELECT * FROM Employees
ORDER BY Salary DESC;
```

Step 11: LIMIT Clause (MySQL)

Fetch top 3 highest paid employees.

```
SELECT * FROM Employees
ORDER BY Salary DESC
LIMIT 3;
```

OUTPUT:



RESULT:

Thus, SQL subqueries, joins, and clauses were implemented successfully using structured SQL queries.