Week 4

4.1) AIM:

Implement SQL queries on a normalized database schema based on the provided schema. For this example: use the schema for a university database, which includes:

- **Students (StudentID, StudentName, Major)**
- **Courses (CourseID, CourseName, Credits)**
- **Enrollments (StudentID, CourseID, EnrollmentDate)**
- **Instructors (InstructorID, InstructorName, Phone)**
- **Course_Instructors (CourseID, InstructorID)**

Description:

University Database Schema

Table Name	Columns
Students	(StudentID, StudentName, Major)
Courses	(CourseID, CourseName, Credits)
Enrollments	(StudentID, CourseID, EnrollmentDate)
Instructors	(InstructorID, InstructorName, Phone)
Course_Instructors	(CourseID, InstructorID)

1. Creating the Tables (DDL Commands)

```
CREATE TABLE Students (
  StudentID INT PRIMARY KEY,
  StudentName VARCHAR(100) NOT NULL,
  Major VARCHAR(50)
);
CREATE TABLE Courses (
  CourseID INT PRIMARY KEY,
  CourseName VARCHAR(100) NOT NULL,
  Credits INT CHECK (Credits > 0)
);
```

```
CREATE TABLE Instructors (
  InstructorID INT PRIMARY KEY,
  InstructorName VARCHAR(100) NOT NULL,
  Phone VARCHAR(15)
);

CREATE TABLE Course_Instructors (
  CourseID INT,
  InstructorID INT,
  PRIMARY KEY (CourseID, InstructorID),
  FOREIGN KEY (CourseID) REFERENCES Courses(CourseID),
  FOREIGN KEY (InstructorID) REFERENCES Instructors(InstructorID)
);

CREATE TABLE Enrollments (
  EnrollmentID INT PRIMARY KEY,
  StudentID INT,
 CourseID INT,
 EnrollmentDate DATE,
  FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
  FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);
```

```
mysql> use test;
Database changed
mysql> CREATE TABLE Students ( StudentID INT PRIMARY KEY,
    -> StudentName VARCHAR(100) NOT NULL, Major VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> CREATE TABLE Courses ( CourseID INT PRIMARY KEY,
    -> CourseName VARCHAR(100) NOT NULL,
    -> Credits INT CHECK (Credits > 0)
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> CREATE TABLE Instructors ( InstructorID INT PRIMARY KEY,
    -> InstructorName VARCHAR(100) NOT NULL, Phone VARCHAR(15)
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> CREATE TABLE Course_Instructors ( CourseID INT,
    -> InstructorID INT,
    -> PRIMARY KEY (CourseID, InstructorID),
    -> FOREIGN KEY (CourseID) REFERENCES Courses(CourseID), FOREIGN KEY (InstructorID) REFERENCES Instructors(InstructorID)
    -> );
Query OK, 0 rows affected (0.05 sec)

mysql>
mysql> CREATE TABLE Enrollments ( EnrollmentID INT PRIMARY KEY, StudentID INT,
    -> CourseID INT, EnrollmentDate DATE,
    -> FOREIGN KEY (StudentID) REFERENCES Students(StudentID), FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
    -> );
Query OK, 0 rows affected (0.09 sec)
```

2. Inserting Sample Records (DML Commands)

```
-- Students
INSERT INTO Students VALUES
(1, 'Asha Patel', 'CSE'),
(2, 'Ravi Kumar', 'ECE'),
(3, 'Maya Singh', 'CSE'),
(4, 'John Doe', 'Mathematics');
-- Courses
INSERT INTO Courses VALUES
(101, 'Database Systems', 4),
(102, 'Operating Systems', 3),
(103, 'Computer Networks', 3),
(104, 'Calculus I', 4);
-- Instructors
INSERT INTO Instructors VALUES
(11, 'Dr. Mehta', '9876543210'),
(12, 'Prof. Sharma', '9123456780'),
(13, 'Dr. Rao', '9000001111');
```

```
mysql> select * from students;
+-----------+-------------+-------------+
| StudentID | StudentName | Major       |
+-----------+-------------+-------------+
|         1 | Asha Patel  | CSE         |
|         2 | Ravi Kumar  | ECE         |
|         3 | Maya Singh  | CSE         |
|         4 | John Doe    | Mathematics |
+-----------+-------------+-------------+
4 rows in set (0.01 sec)

mysql> select * from courses;
+----------+--------------------+---------+
| CourseID | CourseName         | Credits |
+----------+--------------------+---------+
|      101 | Database Systems   |       4 |
|      102 | Operating Systems  |       3 |
|      103 | Computer Networks  |       3 |
|      104 | Calculus I         |       4 |
+----------+--------------------+---------+
4 rows in set (0.00 sec)

mysql> select * from course_instructors;
+----------+--------------+
| CourseID | InstructorID |
+----------+--------------+
|      101 |           11 |
|      103 |           11 |
|      102 |           12 |
|      104 |           13 |
+----------+--------------+
4 rows in set (0.00 sec)

mysql> select * from enrollments;
+--------------+-----------+----------+----------------+
| EnrollmentID | StudentID | CourseID | EnrollmentDate |
+--------------+-----------+----------+----------------+
|         1001 |         1 |      101 | 2025-07-01     |
|         1002 |         1 |      102 | 2025-07-02     |
|         1003 |         2 |      101 | 2025-07-03     |
|         1004 |         3 |      103 | 2025-07-04     |
+--------------+-----------+----------+----------------+
4 rows in set (0.00 sec)
```

```
-- Course_Instructors
INSERT INTO Course_Instructors VALUES
(101, 11),
(102, 12),
(103, 11),
(104, 13);
-- Enrollments
INSERT INTO Enrollments VALUES
(1001, 1, 101, '2025-07-01'),
(1002, 1, 102, '2025-07-02'),
(1003, 2, 101, '2025-07-03'),
(1004, 3, 103, '2025-07-04');
```

3. Important SQL Queries

(a) Display all Students

```
SELECT * FROM Students;
```

Shows all student records.

StudentID	StudentName	Major
1	Asha Patel	CSE
2	Ravi Kumar	ECE
3	Maya Singh	CSE
4	John Doe	Mathematics

(b) Display all Courses having more than 3 Credits

```
SELECT CourseID, CourseName, Credits
FROM Courses
WHERE Credits > 3;
```

CourseID	CourseName	Credits
101	Database Systems	4
104	Calculus I	4

(c) List all Courses with their Instructors

```
SELECT c.CourseName, i.InstructorName
FROM Courses c
JOIN Course_Instructors ci ON c.CourseID = ci.CourseID
JOIN Instructors i ON ci.InstructorID = i.InstructorID;
```

InstructorID	InstructorName
11	Dr. Mehta
12	Prof. Sharma

InstructorID	InstructorName
13	Dr. Gupta

(d) Count number of Courses taught by each Instructor

```
SELECT i.InstructorName, COUNT(ci.CourseID) AS TotalCourses
FROM Instructors i
LEFT JOIN Course_Instructors ci ON i.InstructorID = ci.InstructorID
GROUP BY i.InstructorName;
```

InstructorName	TotalCourses
Dr. Mehta	2
Prof. Sharma	1
Dr. Gupta	1

(e) List all Students enrolled in ‘Database Systems’

```
SELECT s.StudentName, e.EnrollmentDate
FROM Students s
JOIN Enrollments e ON s.StudentID = e.StudentID
JOIN Courses c ON e.CourseID = c.CourseID
WHERE c.CourseName = 'Database Systems';
```

StudentName	EnrollmentDate
Asha Patel	2025-07-01
Ravi Kumar	2025-07-03

(f) Count number of Courses each Student is enrolled in

```
SELECT s.StudentName, COUNT(e.CourseID) AS EnrolledCourses
FROM Students s
LEFT JOIN Enrollments e ON s.StudentID = e.StudentID
GROUP BY s.StudentName;
```

StudentName	EnrolledCourses
Asha Patel	2
Ravi Kumar	1
Maya Singh	1
John Doe	0

(g) List Courses with no Enrollments

```
SELECT c.CourseName
FROM Courses c
LEFT JOIN Enrollments e ON c.CourseID = e.CourseID
WHERE e.EnrollmentID IS NULL;
```

CourseName
Calculus I

(h) Display Students not enrolled in any Course

```
SELECT s.StudentName
FROM Students s
WHERE NOT EXISTS (
  SELECT 1 FROM Enrollments e WHERE e.StudentID = s.StudentID
);
```

StudentName
John Doe

(i) Courses taught by a particular Instructor (e.g., Dr. Mehta)

```
SELECT c.CourseName
FROM Courses c
JOIN Course_Instructors ci ON c.CourseID = ci.CourseID
```

```
JOIN Instructors i ON ci.InstructorID = i.InstructorID
WHERE i.InstructorName = 'Dr. Mehta';
```

CourseName
Database Systems
Computer Networks

(j) Total number of Students enrolled in each Course

```
SELECT c.CourseName, COUNT(e.StudentID) AS TotalStudents
FROM Courses c
LEFT JOIN Enrollments e ON c.CourseID = e.CourseID
GROUP BY c.CourseName;
```

CourseName	TotalStudents
Database Systems	2
Operating Systems	1
Computer Networks	1
Calculus I	0

(k) Create a View showing Student and number of Enrolled Courses

```
CREATE VIEW StudentCourseCount AS
SELECT s.StudentID, s.StudentName, COUNT(e.CourseID) AS TotalCourses
FROM Students s
LEFT JOIN Enrollments e ON s.StudentID = e.StudentID
GROUP BY s.StudentID, s.StudentName;
SELECT * FROM StudentCourseCount;
```

StudentID	StudentName	TotalCourses
1	Asha Patel	2
2	Ravi Kumar	1
3	Maya Singh	1
4	John Doe	0

4.2) AIM:

(A) Implementation of Data Control Language commands – grant and revoke.
(B) Implementation of Transaction Control Language commands – commit, savepoint, and rollback.

Description:

A)

DCL (Data Control Language) in SQL is used to **control access** to data in a database.
It deals with **permissions, privileges, and security levels** of database objects such as tables, views, and sequences.

Command	Purpose
GRANT	Gives privileges (permissions) to users.
REVOKE	Removes previously granted privileges.

GRANT Command

Purpose:

Used to give specific rights to a user or role on database objects.

Syntax:

GRANT privilege_list ON object_name TO user_name [WITH GRANT OPTION];

Program:

Create a New User

Syntax:

CREATE USER username IDENTIFIED BY password;

Example:

CREATE USER student_user IDENTIFIED BY student123;

```
SQL> create user C##student_user IDENTIFIED BY student123
  2  ;

User created.
```

Grant Basic Privileges

To allow the user to log in and create objects, you must grant roles or system privileges.

Grant predefined roles:

GRANT CONNECT, RESOURCE TO student_user;

```
SQL> grant connect,resource to C##student_user;

Grant succeeded.
```

Verify User Creation

To check whether the user is created:

SELECT username, account_status FROM dba_users WHERE username ='STUDENT_USER';

```
SQL> select username,account_status from dba_users where username='student_user';

no rows selected
```

Connect as the New User

Now, you can connect using that account:

SQL> CONNECT student_user/student123;

Grant permission to student_user to read data from the department table.

GRANT SELECT ON department TO student_user;

Now, student_user can execute:

SELECT * FROM system.department;

Grant multiple privileges:

GRANT SELECT, INSERT, UPDATE ON department TO student_user;

Grant multiple privileges:

GRANT SELECT, INSERT, UPDATE ON department TO student_user;

Grant privilege **with grant option** (allows user to pass privilege to others):

GRANT SELECT ON department TO student_user WITH GRANT OPTION;

```
SQL> grant select,insert,update on dept to C##student_user;

Grant succeeded.
```

REVOKE Command

Purpose:

Used to **remove** privileges granted to a user.

Syntax:

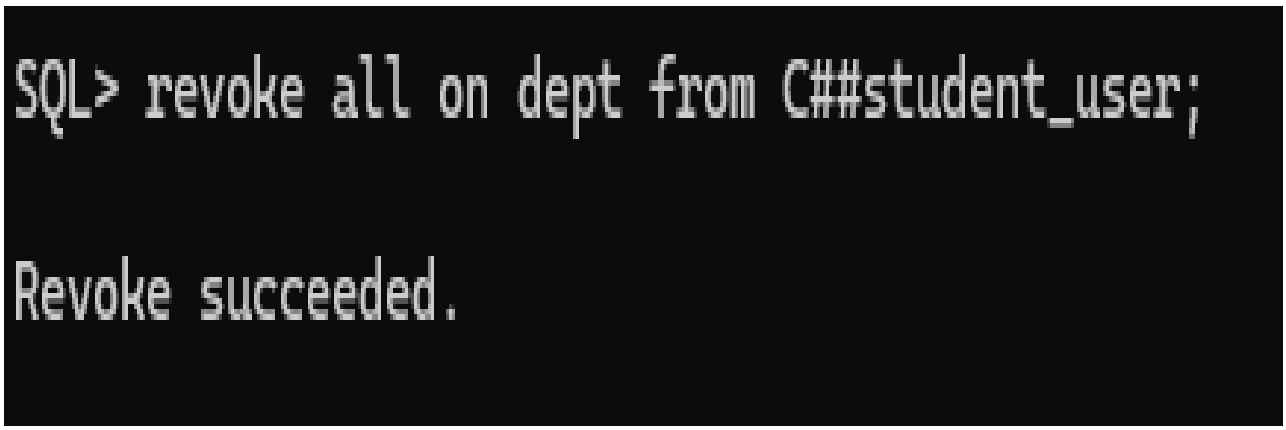
REVOKE privilege_list ON object_name FROM user_name;

Revoke INSERT privilege from student_user:

REVOKE INSERT ON department FROM student_user;

Revoke all privileges:

REVOKE ALL ON department FROM student_user;



B)

Implementation of Transaction Control Language commands - commit, save point, and rollback.

Desciption:

COMMIT saves all **permanent changes** made in the current transaction

SAVEPOINT

Purpose:

- SAVEPOINT allows creating a **checkpoint** in a transaction.
- You can rollback **partially** to this point without affecting previous committed changes.

Example:

SAVEPOINT sp1;

ROLLBACK undoes **all uncommitted changes** or rolls back to a specific **savepoint**.

Rollback to a savepoint

ROLLBACK TO sp1;

Combining TCL Commands

1. Insert multiple rows.
2. Create multiple savepoints.
3. Rollback selectively to any savepoint.
4. Commit final changes to make them permanent.

Summary of TCL Commands

Command	Purpose	Example
COMMIT	Makes all changes in the current transaction permanent	`COMMIT;`
SAVEPOINT	Creates a checkpoint to rollback partially	`SAVEPOINT sp1;`
ROLLBACK	Undoes changes	`ROLLBACK;` (entire transaction)
ROLLBACK TO SAVEPOINT	Undoes changes up to a specific savepoint	`ROLLBACK TO sp1;`

Week 5

AIM: i) Create primary and secondary indexes on a column?

ii) Retrieve data using an indexes?

iii) Insert data and update indexes?

iv) Delete data and impact on indexes? DESCRIPTION:

Description:

Index

Index is a database structure that improves the speed of data retrieval operations on a table at the cost of additional storage space and slower write operations.

Primary Index and Secondary Index

An index on a set of fields that includes the primary key is called primary index and other indexes are called secondary indexes.

A primary index is guaranteed not to contain duplicates, but an index on other fields can contain duplicates. A secondary index contains duplicates.

when indexes speed up reads dramatically, they make DELETE operations substantially more expensive.

To create a secondary index for the table Syntax:

CREATE INDEX index_name ON table_name(columnname) **To drop secondary index Syntax:**

DROP INDEX index_nam

```
SQL> insert into dept values(12,'CSE'),(13,'ECE'),(14,'ME'),(189,'EEE');

4 rows created.

SQL> select *from dept;

   DEPT_ID DEPT_NAME
---------- ----------------------------------------
        12 CSE
        13 ECE
        14 ME
       189 EEE

SQL> create index idx1 on dept(dept_id);

Index created.
```

1:To create Primary Index StudentID:

CREATE TABLE StudentsF (StudentID INT PRIMARY KEY,

FirstName VARCHAR2(50), LastName VARCHAR2(50),

EnrollmentDate DATE

);

Table created. 0.15 seconds

To create Secondary Index LastName:

CREATE INDEX idx_LastName ON StudentsF(LastName); Index created.

0.02 seconds

1. **Insert Data using an index**

INSERT INTO StudentsF VALUES(101, 'John', 'Doe', TO_DATE('15-Aug-2025', 'DD- Mon-YYYY'));

INSERT INTO StudentsF VALUES(102, 'Jane', 'Smith', TO_DATE('16 -Aug-2025', 'DD- Mon-YYYY'));

INSERT INTO StudentsF VALUES(103, 'Ravi', 'Kumar', TO_DATE('25 -Oct-2025', 'DD- Mon-YYYY'));

These rows automatically update both indexes StudentID and LastName.

2. **Retrieve Data Using an Index:**

Using the primary index on StudentID.

SELECT * FROM StudentsF WHERE StudentID = 102;

STUDENTID	FIRSTNAME	LASTNAME	ENROLLMENTDATE
102	Jane	Smith	16-AUG-25

Using the secondary index idx_LastName.

SELECT * FROM StudentsF WHERE LastName = 'Doe';

STUDENTID	FIRSTNAME	LASTNAME	ENROLLMENTDATE
101	John	Doe	15-AUG-25

3. **Delete Data using primary index**

DELETE FROM StudentsF WHERE StudentID = 103; 1 row(s) deleted.

Delete Data using secondary index

DELETE FROM StudentsF WHERE LastName = 'Smith'; 1 row(s) deleted.

View Index Details

desc idx_LastName

Object Type **INDEX** Object **IDX_LASTNAME**

Drop Index

DROP INDEX idx_LastName;

Index dropped.

0.19 seconds Impact on Indexes

Database indexes significantly affect performance

- Indexes dramatically speed up SELECT queries, especially with WHERE clauses, JOIN operations, and ORDER BY statements.
- Instead of scanning entire tables, the database can quickly locate specific rows.