AIM: 4.1 Implement SQL queries on a normalized database schema based on the provided schema

Description:

Normalization is a process of organizing data in a database to reduce redundancy and improve data integrity.

In this experiment, we will use the **University Database Schema** consisting of the following tables:

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

Creating Tables:

```
CREATE TABLE Students (
   StudentID VARCHAR2(10),
   StudentName VARCHAR2(30),
   Major VARCHAR2(30)
);

CREATE TABLE Courses (
   CourseID VARCHAR2(10),
   CourseName VARCHAR2(30),
   Credits NUMBER
);

CREATE TABLE Enrollments (
   StudentID VARCHAR2(10),
   CourseID VARCHAR2(10),
   EnrollmentDate DATE
);

CREATE TABLE Instructors (
   InstructorID VARCHAR2(10),
   InstructorName VARCHAR2(30),
   Phone NUMBER
);

CREATE TABLE Course_Instructors (
   CourseID VARCHAR2(10),
   InstructorID VARCHAR2(10)
);
```

Inserting Sample Data:

```
INSERT INTO Students VALUES ('S01', 'Kiran', 'CSE');
INSERT INTO Students VALUES ('S02', 'Bala', 'ECE');
INSERT INTO Students VALUES ('S03', 'Ravi', 'EEE');

INSERT INTO Courses VALUES ('C01', 'DBMS', 4);
INSERT INTO Courses VALUES ('C02', 'OS', 3);
INSERT INTO Courses VALUES ('C03', 'Networks', 3);

INSERT INTO Enrollments VALUES ('S01', 'C01', '12-SEP-2024');
INSERT INTO Enrollments VALUES ('S02', 'C02', '15-SEP-2024');
INSERT INTO Enrollments VALUES ('S03', 'C03', '20-SEP-2024');

INSERT INTO Instructors VALUES ('I01', 'Suma', 9876543210);
INSERT INTO Instructors VALUES ('I02', 'Raju', 9876501234);

INSERT INTO Course_Instructors VALUES ('C01', 'I01');
INSERT INTO Course_Instructors VALUES ('C02', 'I02');
```

Example Queries:

Display all students and their majors

```
SELECT * FROM Students;
```

STUDENTID	STUDENT NAME	MAJOR
S01	Kiran	CSE
S02	Bala	ECE
S03	Ravi	EEE

Display all courses with credits

```
SELECT CourseName, Credits FROM Courses;
```

ROLLNO	NAME
DBMS	4
OS	3
Networks	3

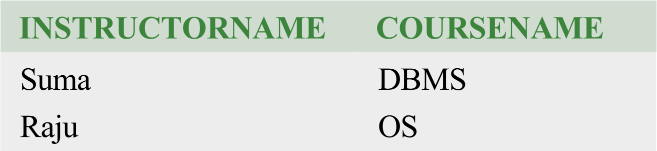
Display students who enrolled in DBMS

```
SELECT s.StudentName, c.CourseName
FROM Students s, Enrollments e, Courses c
WHERE s.StudentID = e.StudentID AND e.CourseID = c.CourseID AND c.CourseName = 'DBMS';
```

STUDENT NAME	COURS ENAME
Kiran	DBMS

Display instructor names along with the courses they teach

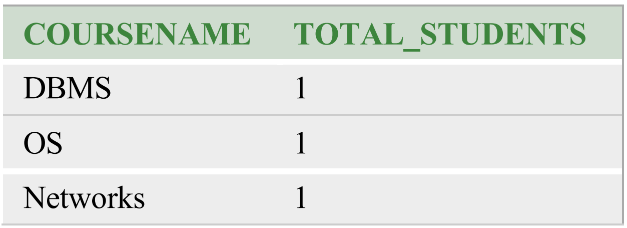
```
SELECT i.InstructorName, c.CourseName
FROM Instructors i, Courses c, Course_Instructors ci
WHERE i.InstructorID = ci.InstructorID AND ci.CourseID = c.CourseID;
```



INSTRUCTORNAME	COURSENAME
Suma	DBMS
Raju	OS

Count number of students enrolled in each course

```
SELECT c.CourseName, COUNT(e.StudentID) AS Total_Students
FROM Courses c, Enrollments e
WHERE c.CourseID = e.CourseID
GROUP BY c.CourseName;
```



COURSENAME	TOTAL_STUDENTS
DBMS	1
OS	1
Networks	1

AIM: 4.2 (A) Implementation of Data Control Language commands — GRANT and REVOKE

Description:

DCL commands control access to data in the database.

- **GRANT:** Allows users to access and manipulate database objects.
- **REVOKE:** Removes previously granted privileges.

Syntax:

```
GRANT privilege_name ON object_name TO user_name;
REVOKE privilege_name ON object_name FROM user_name;
```

Example:

```
CREATE TABLE student_login (
   userid VARCHAR2(10),
   password VARCHAR2(20)
);

GRANT SELECT, INSERT ON student_login TO user1;
REVOKE INSERT ON student_login FROM user1;
```

Explanation:

- The GRANT statement gives user1 permission to **select** and **insert** records into the table.
- The REVOKE statement removes the **insert** permission from user1.

AIM: 4.2 (B) Implementation of Transaction Control Language commands — COMMIT, SAVEPOINT, and ROLLBACK

Description:

TCL commands are used to manage transactions in the database.

- **COMMIT:** Saves all the changes made in the current transaction.
- **ROLLBACK:** Undoes the changes of the current transaction.
- **SAVEPOINT:** Creates a temporary point in a transaction for partial rollback.

Syntax:

```
COMMIT;
ROLLBACK;
SAVEPOINT savepoint_name;
ROLLBACK TO savepoint_name;
```

Example:

```
CREATE TABLE accounts (
   accno NUMBER,
   name VARCHAR2(20),
   balance NUMBER
);

INSERT INTO accounts VALUES (101, 'Ravi', 2000);
SAVEPOINT A;

INSERT INTO accounts VALUES (102, 'Suma', 3000);
SAVEPOINT B;

UPDATE accounts SET balance = balance + 500 WHERE accno = 101;

ROLLBACK TO B;
COMMIT;
```

ACCNO	NAME	BALANCE
101	Ravi	2000
102	Suma	3000

Explanation:

1. **SAVEPOINT A** and **SAVEPOINT B** mark transaction stages.
2. **ROLLBACK TO B** cancels the update done after B but retains all changes before it.
3. **COMMIT** makes the remaining changes permanent.

AIM: 5.1 Create a Primary and Secondary Index on a Column

Description:

An **index** improves the speed of data retrieval operations in a database.
There are two main types:

- **Primary Index** – created automatically when a primary key is defined.
- **Secondary Index** – created manually on non-primary key columns to improve search performance.

Indexes work like book indexes — instead of scanning all pages (rows), the database can jump directly to the desired data.

Syntax:

```
-- Primary Index (Automatically created using PRIMARY KEY)
CREATE TABLE table_name (
  column1 datatype PRIMARY KEY,
  column2 datatype,
  ...
);

-- Secondary Index (Manually created)
CREATE INDEX index_name
ON table_name (column_name);
```

Example:

```
CREATE TABLE Students (
  StudentID VARCHAR2(10) PRIMARY KEY,
  StudentName VARCHAR2(30),
  Major VARCHAR2(20)
);

-- Creating Secondary Index on StudentName
CREATE INDEX idx_studentname
ON Students (StudentName);
```

Output:

```
Table created.
Index created.
```

AIM: 5.2 Retrieve Data Using an Index

Description:

When a query uses an indexed column in the WHERE clause, the database engine uses the index to quickly locate the matching rows, improving performance.

Example:

```
SELECT * FROM Students WHERE StudentName = 'Kiran';
```

Since the StudentName column has a secondary index (idx_studentname), this query will retrieve the record faster than a full table scan.

StudentID	StudentName	Major
S01	Kiran	CSE

AIM: 5.3 Insert Data and Update Indexes

Description:

When new records are inserted, the indexes are automatically updated by the database. This ensures that all future retrievals remain efficient.

Example:

```
INSERT INTO Students VALUES ('S02', 'Ravi', 'ECE');
INSERT INTO Students VALUES ('S03', 'Bala', 'EEE');
```

The database automatically updates:

- The **Primary Index** for StudentID
- The **Secondary Index** for StudentName

To Verify:

```
SELECT * FROM Students;
```

STUDENTID	STUDENT NAME	MAJOR
S01	Kiran	CSE
S02	Ravi	ECE
S03	Bala	EEE

AIM: 5.4 Delete Data and Observe Impact on Indexes

Description:

When a row is deleted, the corresponding entries in all indexes are also automatically removed by the database.
This maintains data consistency and prevents invalid index references.

Example:

```
DELETE FROM Students WHERE StudentID = 'S02';
```

The index entries for 'S02' and 'Ravi' are automatically deleted from the primary and secondary indexes.

To Verify:

```
SELECT * FROM Students;
```

STUDENTID	STUDENT NAME	MAJOR
S01	Kiran	CSE
S03	Bala	ECE

Conclusion:

- Primary indexes are automatically created and maintained by the DBMS.
- Secondary indexes are created manually for faster access on non-key columns.
- Both indexes automatically update when data is inserted, deleted, or modified.