AIM: 4.1 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)

DESCRIPTION:

Query

A query is a request for data or information from a database table or combination of tables.
SELECT SQL Query is used to retrieve data from a table.

1. Students Table
Purpose: Stores student information
Columns:
StudentID (Primary Key) - Unique identifier for each student
StudentName - Name of the student
Major - Academic major/field of study

2. Courses Table
Purpose: Contains course catalog information
Columns:
CourseID (Primary Key) - Unique identifier for each course
CourseName - Name of the course
Credits - Number of credit hours for the course

3. Enrollments Table
Purpose: Records which students are enrolled in which courses (Many-to-Many relationship)
Columns:
StudentID (Foreign Key) - References Students table
CourseID (Foreign Key) - References Courses table
EnrollmentDate - Date when student enrolled in the course

4. Instructors Table
Purpose: Stores instructor/professor information
Columns:
InstructorID (Primary Key) - Unique identifier for each instructor
InstructorName - Name of the instructor
Phone - Contact phone number

5. Course_Instructors Table
Purpose: Maps which instructors teach which courses (Many-to-Many relationship)
Columns:
CourseID (Foreign Key) - References Courses table
InstructorID (Foreign Key) - References Instructors table

Step 1: Create Tables

Creating students table

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

Table created.

Creating courses table

```
CREATE TABLE Courses (
    CourseID INTEGER PRIMARY KEY,
    CourseName VARCHAR2(30),
    Credits INTEGER
);
```

Table created.

Creating enrollments table

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

Table created.

Creating instructors table

```
CREATE TABLE Instructors (
    InstructorID INTEGER PRIMARY KEY,
    InstructorName VARCHAR2(30),
    Phone INTEGER
);
```

Table created.

Creating course_instructors table

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

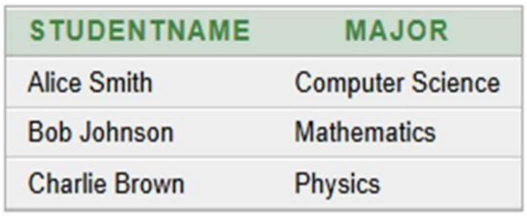
```
Table created.
```

Step 2: Inserting data into tables
This step populates the tables with the provided sample data. The output is a confirmation of the number of rows inserted.

Query:
```
INSERT INTO Students VALUES (1, 'Alice Smith', 'Computer Science');
INSERT INTO Students VALUES (2, 'Bob Johnson', 'Mathematics');
INSERT INTO Students VALUES (3, 'Charlie Brown', 'Physics');
```
Query:
```
INSERT INTO Courses VALUES (101, 'Introduction to Programming', 3);
INSERT INTO Courses VALUES (102, 'Calculus I', 4);
INSERT INTO Courses VALUES (103, 'Classical Mechanics', 3);
```
Query:
```
INSERT INTO Enrollments VALUES (1, 101, '01-SEP-24');
INSERT INTO Enrollments VALUES (1, 102, '01-SEP-24');
INSERT INTO Enrollments VALUES (2, 102, '01-SEP-24');
INSERT INTO Enrollments VALUES (3, 103, '01-SEP-24');
```
Query:
```
INSERT INTO Instructors VALUES (1001, 'Dr. Emily White', 7225148456);
INSERT INTO Instructors VALUES (1002, 'Prof. David Green', 9822663346);
```
Query:
```
INSERT INTO Course_Instructors VALUES (101, 1001);
INSERT INTO Course_Instructors VALUES (102, 1002);
INSERT INTO Course_Instructors VALUES (103, 1002);
```

Step 3: Execute Retrieval Queries

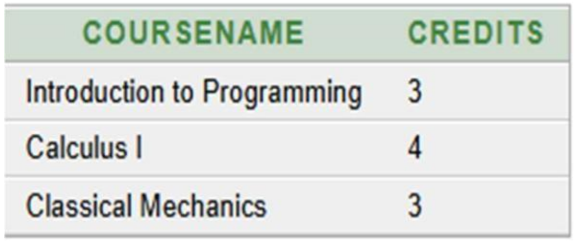
Retrieve all students and their majors
select StudentName,Major from Students;



STUDENTNAME	MAJOR
Alice Smith	Computer Science
Bob Johnson	Mathematics
Charlie Brown	Physics

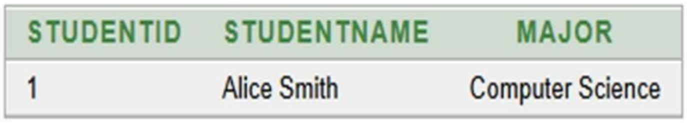
Queries:

List all courses along with the number of credits.
select CourseName, Credits from Courses;



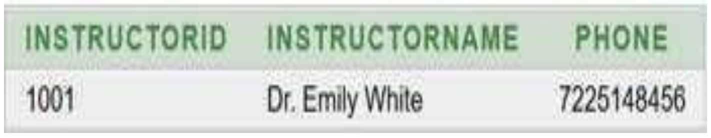
COURSENAME	CREDITS
Introduction to Programming	3
Calculus I	4
Classical Mechanics	3

Find all students enrolled in a specific course (e.g., 'Introduction to Programming').
SELECT s.* FROM students s
JOIN enrollments e ON s.studentID = e.studentID
JOIN courses c ON e.courseID = c.courseID
WHERE c.coursename = 'Introduction to Programming';



STUDENTID	STUDENTNAME	MAJOR
1	Alice Smith	Computer Science

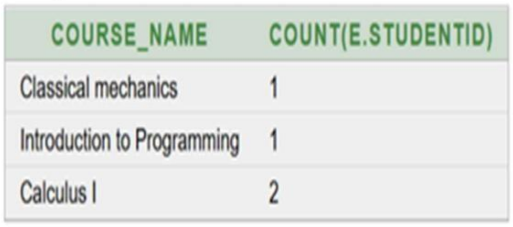
Get the list of instructors teaching a specific course(e.g., 'Introduction to Programming')
SELECT i.* FROM Instructors i, Course_Instructors ci, Courses c
WHERE i.InstructorID = ci.InstructorID
AND ci.CourseID = c.CourseID
AND c.CourseName = 'Introduction to Programming';



INSTRUCTORID	INSTRUCTORNAME	PHONE
1001	Dr. Emily White	7225148456

Count the number of students enrolled in each course.

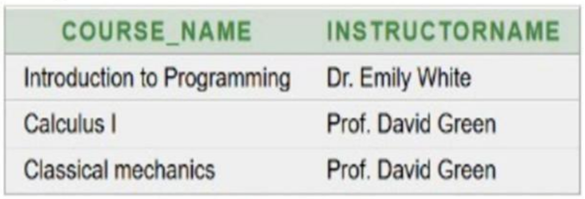
```
SELECT c.CourseName, COUNT(e.StudentID) FROM
Courses c, Enrollments e
WHERE c.CourseID = e.CourseID
GROUP BY c.CourseName;
```



COURSE_NAME	COUNT(E.STUDENTID)
Classical mechanics	1
Introduction to Programming	1
Calculus I	2

List courses along with their instructor names

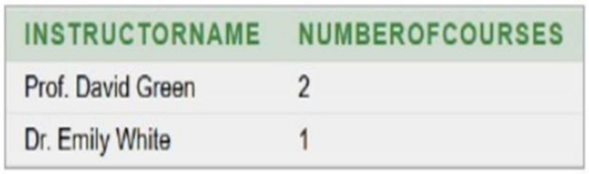
```
SELECT c.CourseName, i.Instructorname
FROM Courses c, Course_Instructors ci, Instructors i WHERE c.CourseID
= ci.CourseID
AND ci.InstructorID = i.InstructorID;
```



COURSE_NAME	INSTRUCTORNAME
Introduction to Programming	Dr. Emily White
Calculus I	Prof. David Green
Classical mechanics	Prof. David Green

Get the number of courses taught by each instructor

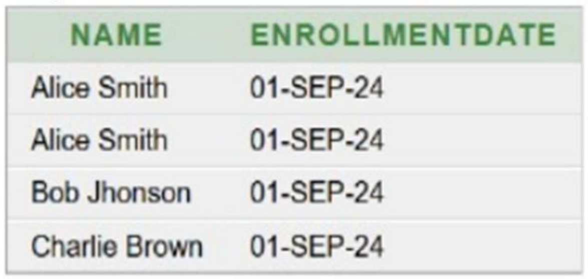
```
SELECT i.Instructorname, COUNT(ci.CourseID) AS NumberOfCourses FROM
Instructors i, Course_Instructors ci
WHERE i.InstructorID = ci.InstructorID GROUP BY
i.Instructorname;
```



INSTRUCTORNAME	NUMBEROFCOURSES
Prof. David Green	2
Dr. Emily White	1

Find students enrolled after a certain date (e.g., '2024-01-01')

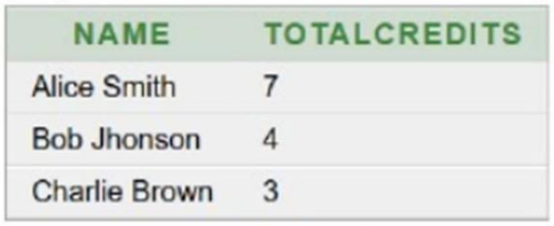
```
SELECT s.StudentName, e.EnrollmentDate
FROM Students s, Enrollments e
WHERE s.StudentID = e.StudentID
AND e.EnrollmentDate > TO_DATE('2024-01-01','YYYY-MM-DD');
```



NAME	ENROLLMENTDATE
Alice Smith	01-SEP-24
Alice Smith	01-SEP-24
Bob Jhonson	01-SEP-24
Charlie Brown	01-SEP-24

Show each student with the total number of credits they are taking

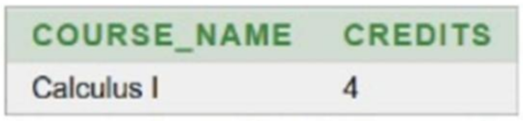
```
SELECT s.StudentName, SUM(c.credits) AS TotalCredits FROM
Students s, Enrollments e, Courses c
WHERE s.StudentID = e.StudentID
AND e.CourseID = c.CourseID
GROUP BY s.StudentName;
```



NAME	TOTALCREDITS
Alice Smith	7
Bob Jhonson	4
Charlie Brown	3

Show courses that have more than 3 credits.

```
SELECT c.CourseName, c.credits FROM Courses c WHERE
c.credits > 3;
```



COURSE_NAME	CREDITS
Calculus I	4

<u>AIM</u>: 4.2 (A) Implementation of Data Control Language commands – grant and revoke

<u>DESCRIPTION:</u>

1. CREATE USER Purpose:

Creates a new database user account with login credentials.

Syntax:

```
CREATE USER username IDENTIFIED BY 'password';
```

2. GRANT

Purpose: Provides specific privileges or permissions to database users.

Syntax:

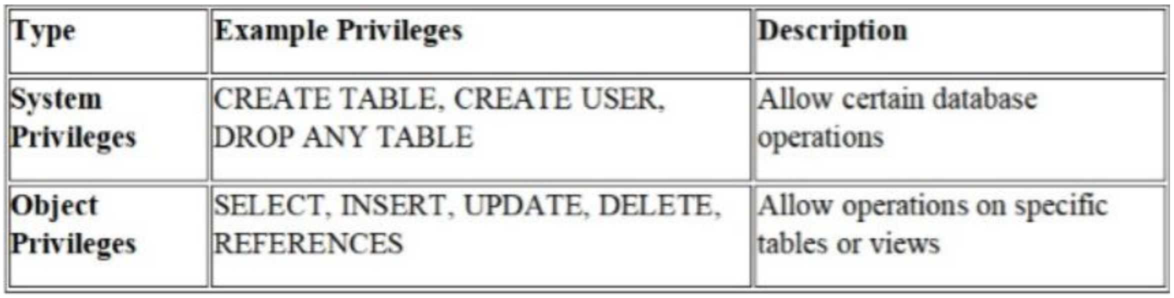
```
GRANT privilege_name ON object_name TO user_name [WITH GRANT OPTION];
```

3. REVOKE

Purpose: Removes previously granted privileges from database users.

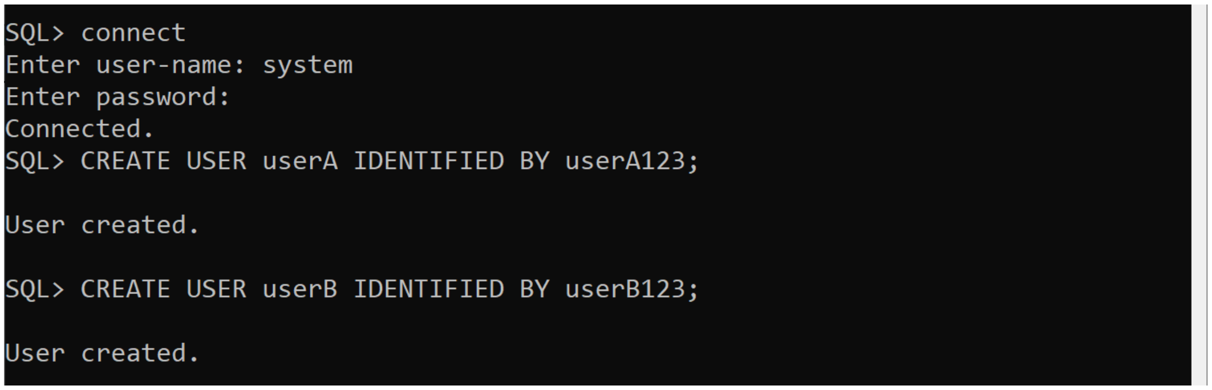
Syntax:

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



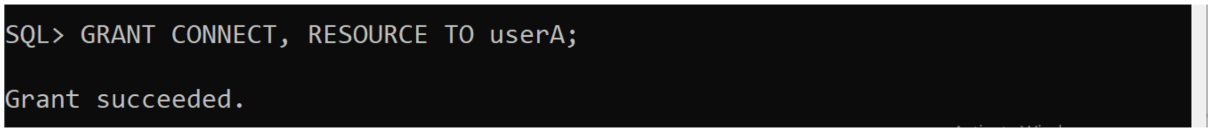
Type	Example Privileges	Description
System Privileges	CREATE TABLE, CREATE USER, DROP ANY TABLE	Allow certain database operations
Object Privileges	SELECT, INSERT, UPDATE, DELETE, REFERENCES	Allow operations on specific tables or views

Creating two users



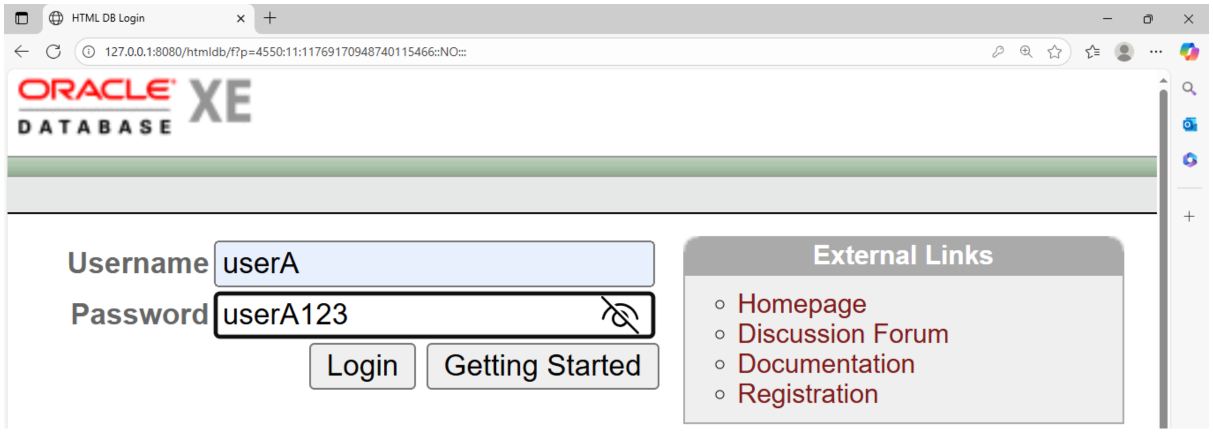
```
SQL> connect
Enter user-name: system
Enter password:
Connected.
SQL> CREATE USER userA IDENTIFIED BY userA123;

User created.

SQL> CREATE USER userB IDENTIFIED BY userB123;

User created.
```

Grant them minimal access to connect and create objects:



```
SQL> GRANT CONNECT, RESOURCE TO userA;

Grant succeeded.
```

Login as userA



HTML DB Login
127.0.0.1:8080/htmldb/f?p=4550:11:11769170948740115466::NO:::
ORACLE DATABASE XE

Username userA

Password userA123

Create emp table

```
CREATE TABLE emp (
  emp_id NUMBER PRIMARY KEY,
  emp_name VARCHAR2(50),
  salary NUMBER
);
Table created.
```

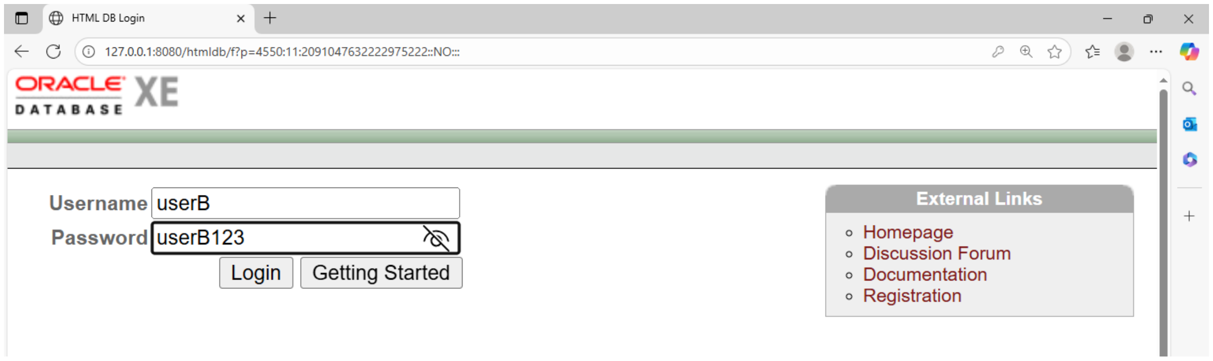
Inserting rows in emp table

```
INSERT INTO emp VALUES (101, 'Alice', 50000);
1 row(s) inserted.


INSERT INTO emp VALUES (102, 'Bob', 60000);
1 row(s) inserted.


Commit;
Commit Complete.
```

userA grants SELECT privilege on this table to userB:

```
GRANT SELECT ON emp TO userB;
Statement processed.
```

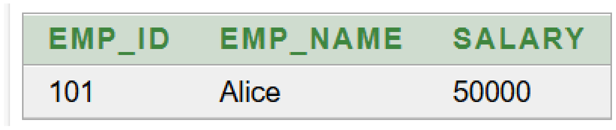
Login as userB



Username	userB
Password	userB123

Displaying emp table of userA

SELECT * FROM userA.emp;



EMP_ID	EMP_NAME	SALARY
101	Alice	50000

Granting Multiple Privileges

Login as userA and type

GRANT SELECT, INSERT, UPDATE ON emp TO userB;

Statement processed.

Login as userB and type

INSERT INTO userA.emp VALUES (103, 'Charlie', 70000);

1 row(s) inserted.

Revoking Privileges

Login as userA

```
REVOKE SELECT, INSERT, UPDATE ON emp FROM userB;
```

Result:
userB can no longer query or modify userA.emp.

Drop user

```
Drop user userB;
```

```
User Dropped.
```

```
Drop user userA cascade;
```

```
User Dropped.
```

<u>AIM</u>: 4.2 (B) Implementation of Transaction Control Language commands – commit, save point and rollback

<u>DESCRIPTION:</u>

Transaction Control Language (TCL)

TCL commands are used to manage transactions in a database.

A transaction is a sequence of SQL operations performed as a single logical unit of work.

TCL commands ensure data integrity and consistency.

1. COMMIT

Commits (saves) all the changes made by a transaction permanently to the database.

After COMMIT, changes cannot be undone.

Syntax:

```
COMMIT;
```

2. SAVEPOINT

Creates a checkpoint (marker) within a transaction.

Allows partial rollback to that point without affecting the entire transaction.

Multiple savepoints can be created in a single transaction.

Syntax:

```
SAVEPOINT savepoint_name;
```

3. ROLLBACK

Rolls back (undoes) all changes made in the current transaction (if no COMMIT has been issued).

Can also rollback partially to a specific SAVEPOINT.

Syntax:

```
ROLLBACK;     -- Undo all uncommitted changes

ROLLBACK TO savepoint_name; -- Undo changes after the specified savepoint
```

Program:

Creating table

```
create table std1(rollno integer,name varchar2(20),branch varchar2(20))
```

```
Table created
```

Inserting rows

```
insert into std1 values (201,'ramesh','mech');
```

```
1 row(s) inserted.
```

Creating Save Point A

```
begin

SAVEPOINT A;

end;
```

```
statement processed.
```

Inserting rows

```
insert into std1 values(202,'geetha','civil')
```

```
1 row(s) inserted.
```

Creating Save Point B

```
begin

SAVEPOINT B;

end;
```

```
statement processed.
```

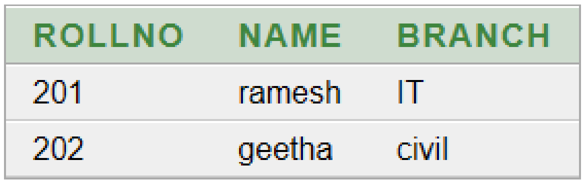
Updating a row

update std1 SET branch='IT' where rollno=201;

1 row(s) updated.

Display table rows

select * from std1;



ROLLNO	NAME	BRANCH
201	ramesh	IT
202	geetha	civil

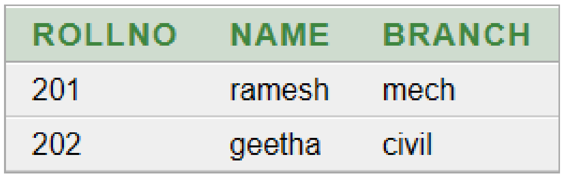
Rollback to savepoint B

ROLLBACK TO B;

Statement processed.

Display table rows

select * from std1;



ROLLNO	NAME	BRANCH
201	ramesh	mech
202	geetha	civil

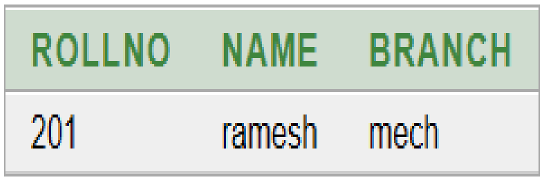
Rollback to savepoint A

ROLLBACK TO A

Statement processed.

Display table rows

select * from std1;



ROLLNO	NAME	BRANCH
201	ramesh	mech