Experiment-3

1)Aim->

To design and implement a normalized relational database schema for managing students, courses, and enrollment details with grades; ensure referential integrity using primary and foreign keys; insert sample data for simulation; perform transaction control operations such as BEGIN, SAVEPOINT, and ROLLBACK; and retrieve comprehensive student-course-grade reports through SQL joins.

2) Objective->

· **Stores and organizes** student personal details, course information, and enrollment records in a relational format.

Maintains referential integrity by using appropriate primary key and foreign key constraints between related tables.

Facilitates accurate data entry by enforcing valid data types and constraints (e.g., proper grade formats, meaningful names).

Supports controlled data operations through the use of SQL transaction management commands such as BEGIN, SAVEPOINT, ROLLBACK, and COMMIT.

Simulates real-world academic scenarios, including enrolling students in multiple courses and assigning grades.

Provides efficient data retrieval by writing SQL join queries to generate consolidated reports showing each student's enrolled courses and grades.

Ensures data reliability and scalability, allowing for future expansion of courses, students, and grading systems without compromising database performance or consistency.

3) Procedure/Algorithum->

· **Start**

· **Identify Entities & Attributes**

Define Students(student_id, name, dob)

Define Courses(course_id, title)

Define Enrollments(enroll_id, student_id, course_id, grade)

•

· **Design Schema**

Set primary keys for each table

Set foreign keys in Enrollments referencing Students and Courses.

· **Create Tables**

Execute SQL CREATE TABLE statements with constraints.

· **Insert Data**

Execute INSERT INTO Students for sample student records.

Execute INSERT INTO Courses for sample course records.

Execute INSERT INTO Enrollments for sample enrollment records.

· **Begin Transaction**

Use BEGIN to start a transaction.

· **Enroll Student**

Insert a new record into Enrollments for a selected student and course.

· **Set Savepoint**

Use SAVEPOINT before the next critical enrollment.

· **Perform Additional Operation**

Insert more enrollments.

If an error occurs, ROLLBACK TO SAVEPOINT.

· **Commit Changes**

If all operations succeed, COMMIT to save permanently.

· **Generate Report**

Use SQL JOIN queries to retrieve student_name, course_title, and grade.

· **Display Results**

Output the joined data in the required tabular format.

· **End**

4) Problem statement->

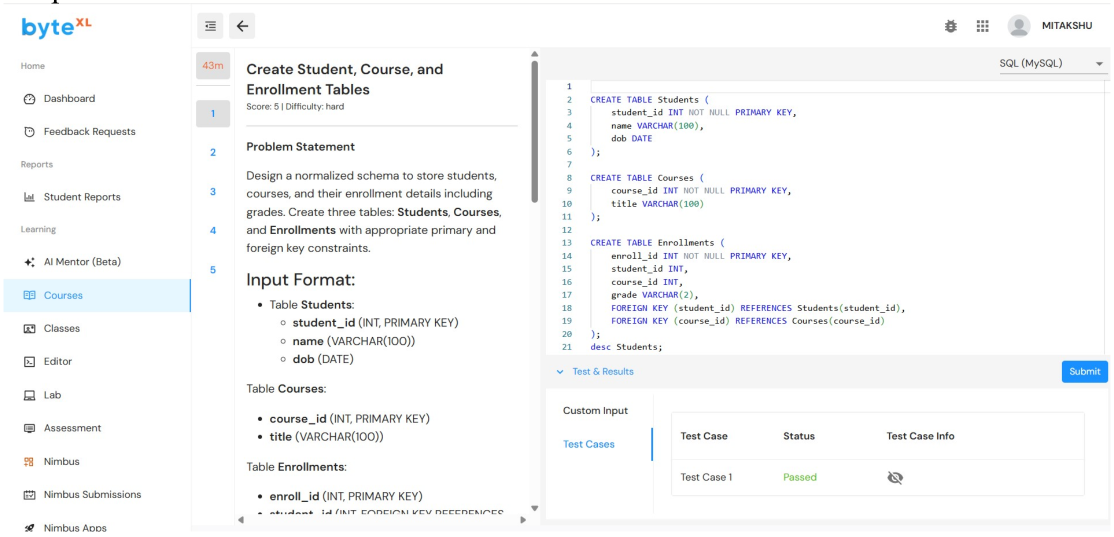
Problem statement-1

Design and implement a **normalized relational database schema** to manage student records, course details, and their enrollment information including grades. The system should maintain data integrity through the use of **primary keys** and **foreign keys**.

Query-1

```sql
CREATE TABLE Students (
    student_id INT NOT NULL PRIMARY KEY,
    name VARCHAR(100),
    dob DATE
);
CREATE TABLE Courses (
    course_id INT NOT NULL PRIMARY KEY,
    title VARCHAR(100)
);
CREATE TABLE Enrollments (
    enroll_id INT NOT NULL PRIMARY KEY,
    student_id INT,
    course_id INT,
    grade VARCHAR(2),
    FOREIGN KEY (student_id) REFERENCES Students(student_id),
    FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);
desc Students;
desc Courses;
desc Enrollments;
```

Output-1



Create Student, Course, and Enrollment Tables
Score: 5 | Difficulty: hard

Problem Statement

Design a normalized schema to store students, courses, and their enrollment details including grades. Create three tables: **Students**, **Courses**, and **Enrollments** with appropriate primary and foreign key constraints.

Input Format:

- Table **Students**:
  - **student_id** (INT, PRIMARY KEY)
  - **name** (VARCHAR(100))
  - **dob** (DATE)

Table **Courses**:

- **course_id** (INT, PRIMARY KEY)
- **title** (VARCHAR(100))

Table **Enrollments**:

- **enroll_id** (INT, PRIMARY KEY)

```sql

CREATE TABLE Students (
    student_id INT NOT NULL PRIMARY KEY,
    name VARCHAR(100),
    dob DATE
);

CREATE TABLE Courses (
    course_id INT NOT NULL PRIMARY KEY,
    title VARCHAR(100)
);

CREATE TABLE Enrollments (
    enroll_id INT NOT NULL PRIMARY KEY,
    student_id INT,
    course_id INT,
    grade VARCHAR(2),
    FOREIGN KEY (student_id) REFERENCES Students(student_id),
    FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);
desc Students;
```

Test Case	Status	Test Case Info
Test Case 1	Passed	

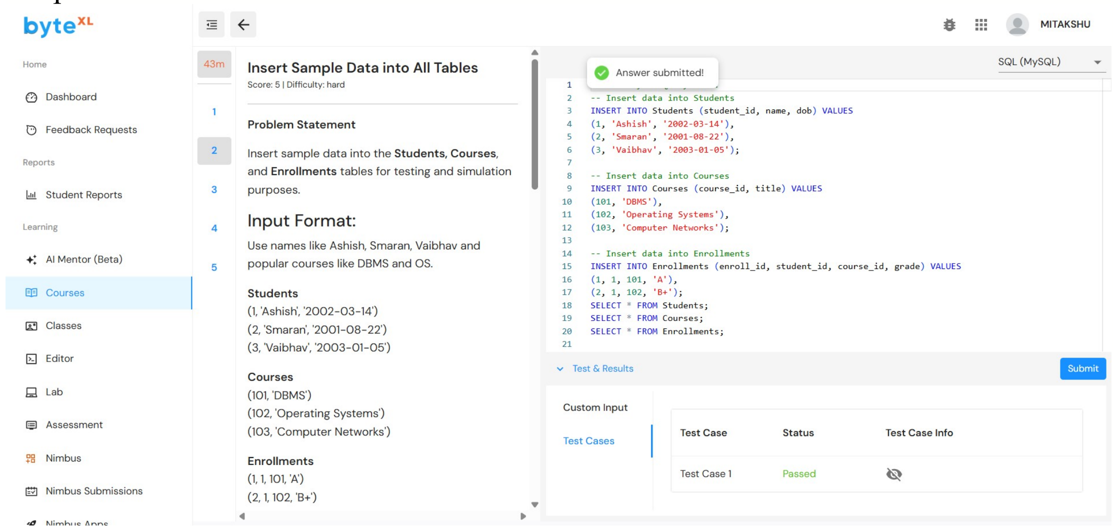
Problem statement-2

Insert meaningful sample records into the Students, Courses, and Enrollments tables to test the previously designed normalized schema and simulate realistic enrollment scenarios. Ensure all inserted data respects the defined constraints and maintains referential integrity.

Query-2

```
INSERT INTO Students (student_id, name, dob) VALUES
(1, 'Ashish', '2002-03-14'),
(2, 'Smaran', '2001-08-22'),
(3, 'Vaibhav', '2003-01-05');
-- Insert data into Courses
INSERT INTO Courses (course_id, title) VALUES
(101, 'DBMS'),
(102, 'Operating Systems'),
(103, 'Computer Networks');
-- Insert data into Enrollments
INSERT INTO Enrollments (enroll_id, student_id, course_id, grade) VALUES
(1, 1, 101, 'A'),
(2, 1, 102, 'B+');
SELECT * FROM Students;
SELECT * FROM Courses;
SELECT * FROM Enrollments;
```

Output-2



Insert Sample Data into All Tables

Score: 5 | Difficulty: hard

Problem Statement

Insert sample data into the **Students, Courses,** and **Enrollments** tables for testing and simulation purposes.

Input Format:

Use names like Ashish, Smaran, Vaibhav and popular courses like DBMS and OS.

Students
(1, 'Ashish', '2002-03-14')
(2, 'Smaran', '2001-08-22')
(3, 'Vaibhav', '2003-01-05')

Courses
(101, 'DBMS')
(102, 'Operating Systems')
(103, 'Computer Networks')

Enrollments
(1, 1, 101, 'A')
(2, 1, 102, 'B+')

```
-- Insert data into Students
INSERT INTO Students (student_id, name, dob) VALUES
(1, 'Ashish', '2002-03-14'),
(2, 'Smaran', '2001-08-22'),
(3, 'Vaibhav', '2003-01-05');

-- Insert data into Courses
INSERT INTO Courses (course_id, title) VALUES
(101, 'DBMS'),
(102, 'Operating Systems'),
(103, 'Computer Networks');

-- Insert data into Enrollments
INSERT INTO Enrollments (enroll_id, student_id, course_id, grade) VALUES
(1, 1, 101, 'A'),
(2, 1, 102, 'B+');
SELECT * FROM Students;
SELECT * FROM Courses;
SELECT * FROM Enrollments;
```

Test Case	Status	Test Case Info
Test Case 1	Passed	

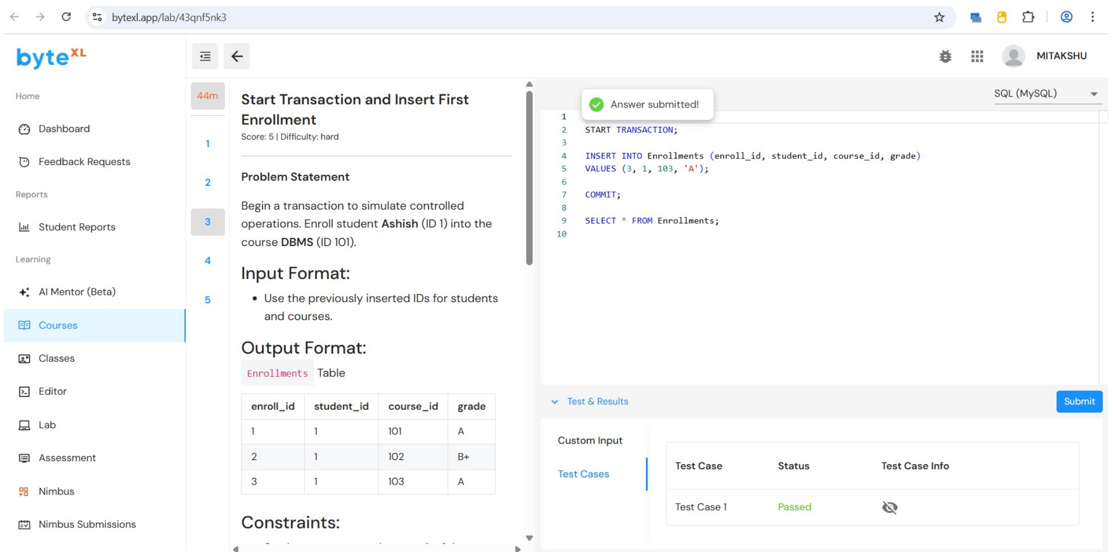
Problem statement-3

Simulate a **controlled database operation** by beginning a transaction to enroll an existing student into a new course. Specifically, enroll **student Ashish (ID 1)** into the course **Computer Networks (ID 103)** with a valid grade. Ensure that all changes are made within a transaction so they can be committed or rolled back as needed.

Query-3

```
START TRANSACTION;
INSERT INTO Enrollments (enroll_id, student_id, course_id, grade)
VALUES (3, 1, 103, 'A');
COMMIT;
SELECT * FROM Enrollments;
```

Output-3



Start Transaction and Insert First Enrollment
Score: 5 | Difficulty: hard

Problem Statement

Begin a transaction to simulate controlled operations. Enroll student **Ashish** (ID 1) into the course **DBMS** (ID 101).

Input Format:

- Use the previously inserted IDs for students and courses.

Output Format:

`Enrollments` Table

enroll_id	student_id	course_id	grade
1	1	101	A
2	1	102	B+
3	1	103	A

Constraints:

Answer submitted!

SQL (MySQL)

```sql

START TRANSACTION;

INSERT INTO Enrollments (enroll_id, student_id, course_id, grade)
VALUES (3, 1, 103, 'A');

COMMIT;

SELECT * FROM Enrollments;

```

Test Case	Status	Test Case Info
Test Case 1	Passed	

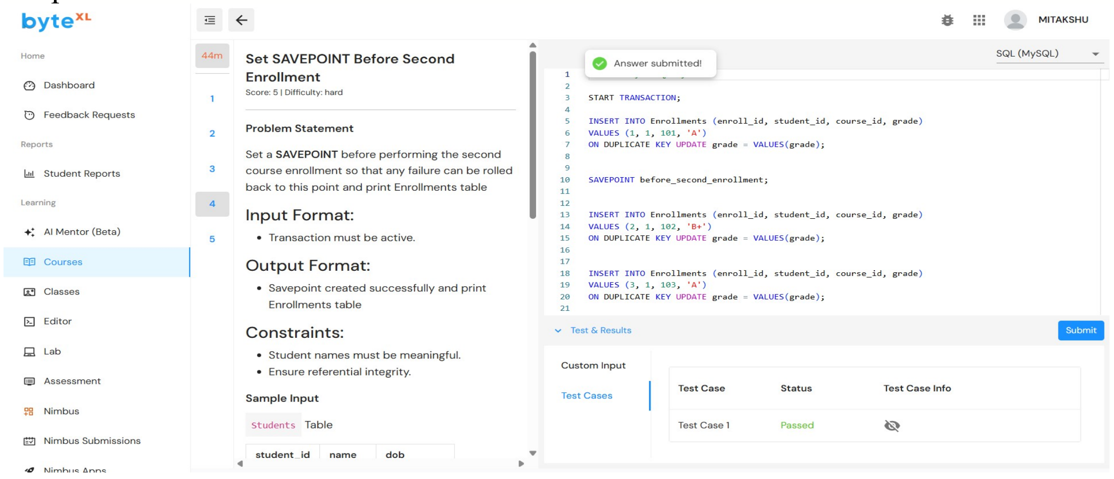
Problem statement-4

Within an **active transaction**, create a **SAVEPOINT** before performing a second course enrollment for a student so that any failure occurring afterward can be rolled back to this point. After creating the savepoint, display the current contents of the Enrollments table.

Query-4

```sql
START TRANSACTION;
INSERT INTO Enrollments (enroll_id, student_id, course_id, grade)
VALUES (1, 1, 101, 'A')
ON DUPLICATE KEY UPDATE grade = VALUES(grade);
SAVEPOINT before_second_enrollment;
INSERT INTO Enrollments (enroll_id, student_id, course_id, grade)
VALUES (2, 1, 102, 'B+')
ON DUPLICATE KEY UPDATE grade = VALUES(grade);
INSERT INTO Enrollments (enroll_id, student_id, course_id, grade)
VALUES (3, 1, 103, 'A')
ON DUPLICATE KEY UPDATE grade = VALUES(grade);
COMMIT;
SELECT * FROM Enrollments;
```

Output-4



Set SAVEPOINT Before Second Enrollment
Score: 5 | Difficulty: hard

Problem Statement

Set a **SAVEPOINT** before performing the second course enrollment so that any failure can be rolled back to this point and print Enrollments table

Input Format:

- Transaction must be active.

Output Format:

- Savepoint created successfully and print Enrollments table

Constraints:

- Student names must be meaningful.
- Ensure referential integrity.

Sample Input

`Students` Table

student_id	name	dob

Answer submitted!

SQL (MySQL)

```sql


START TRANSACTION;

INSERT INTO Enrollments (enroll_id, student_id, course_id, grade)
VALUES (1, 1, 101, 'A')
ON DUPLICATE KEY UPDATE grade = VALUES(grade);


SAVEPOINT before_second_enrollment;


INSERT INTO Enrollments (enroll_id, student_id, course_id, grade)
VALUES (2, 1, 102, 'B+')
ON DUPLICATE KEY UPDATE grade = VALUES(grade);


INSERT INTO Enrollments (enroll_id, student_id, course_id, grade)
VALUES (3, 1, 103, 'A')
ON DUPLICATE KEY UPDATE grade = VALUES(grade);

```

Test Case	Status	Test Case Info
Test Case 1	Passed	

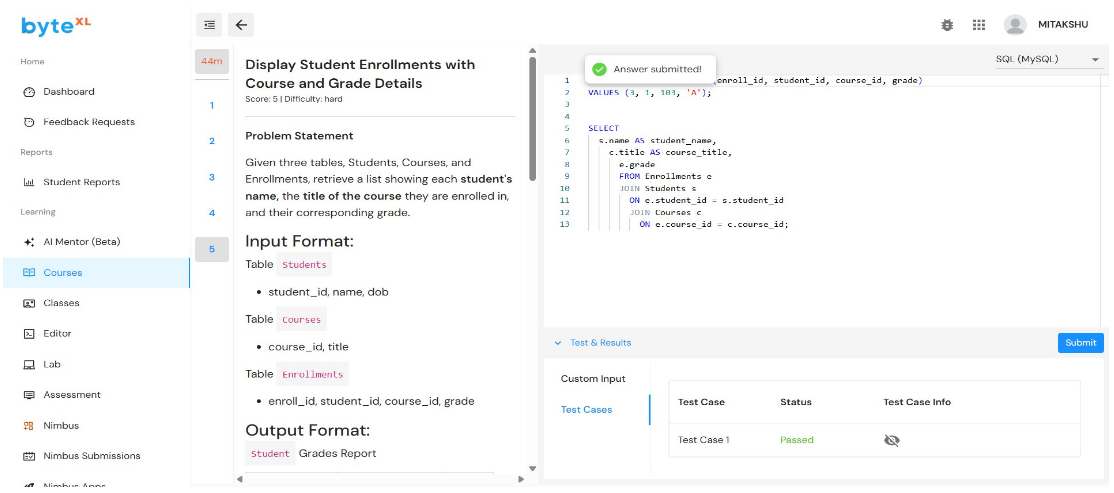
Problem statement-5

Retrieve a **student grades report** by combining information from three tables — Students, Courses, and Enrollments. The report should display each student's name, the title of the course they are enrolled in, and their corresponding grade. The query must ensure that only valid student-course relationships are shown by enforcing the existing foreign key constraints.

Query-5

```sql
INSERT INTO Enrollments (enroll_id, student_id, course_id, grade)
VALUES (3, 1, 103, 'A');
SELECT
  s.name AS student_name,
    c.title AS course_title,
      e.grade
      FROM Enrollments e
      JOIN Students s
        ON e.student_id = s.student_id
        JOIN Courses c
          ON e.course_id = c.course_id;
```

Output-5



Display Student Enrollments with Course and Grade Details

Score: 5 | Difficulty: hard

Problem Statement

Given three tables, Students, Courses, and Enrollments, retrieve a list showing each **student's name**, the **title of the course** they are enrolled in, and their corresponding grade.

Input Format:

Table `Students`

- student_id, name, dob

Table `Courses`

- course_id, title

Table `Enrollments`

- enroll_id, student_id, course_id, grade

Output Format:

`Student` Grades Report

Answer submitted!

```sql
enroll_id, student_id, course_id, grade)
VALUES (3, 1, 103, 'A');


SELECT
  s.name AS student_name,
    c.title AS course_title,
      e.grade
      FROM Enrollments e
      JOIN Students s
        ON e.student_id = s.student_id
        JOIN Courses c
          ON e.course_id = c.course_id;
```

Test Case	Status	Test Case Info
Test Case 1	Passed	

5)Learning outcomes->

Database Design Skills – Ability to design a normalized relational database schema with appropriate primary and foreign key constraints.

Data Manipulation Proficiency – Skill in inserting, updating, and retrieving data using SQL `INSERT`, `UPDATE`, and `SELECT` statements.

Transaction Management – Understanding of transaction control commands (`BEGIN`, `SAVEPOINT`, `ROLLBACK`, `COMMIT`) to ensure controlled and reliable database operations.

SQL Joins & Querying – Proficiency in using `INNER JOIN` and other SQL joins to combine data from multiple related tables into meaningful reports.