DBT Project

Backend Development: Team 13

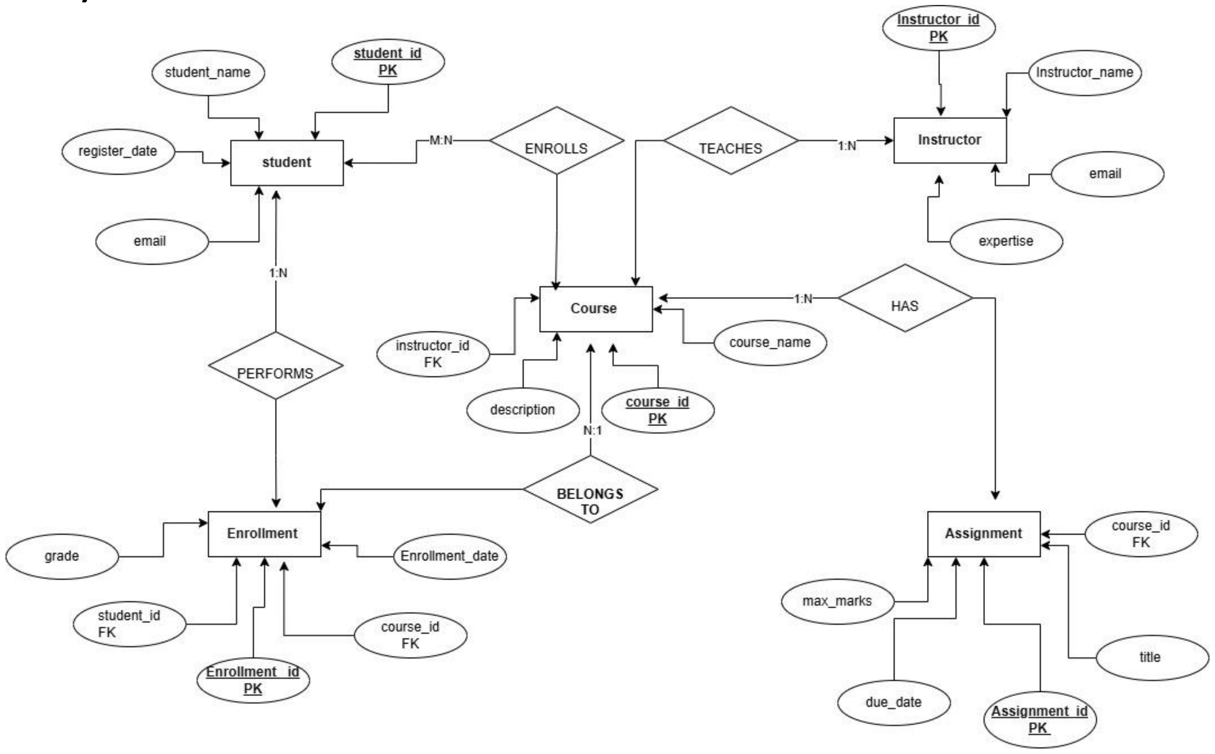
Akanksha Jeevan Puri – 250840320013

Madhuri Kedarnath Chavan – 250840320093

Topic: Learning Management System (LMS)

1.Draw an ER Diagram in draw.io showing entities, attributes, and relationships.

- **Identify all major entities, their attributes, and primary keys.**
- **Show relationships (1–M, M–N, 1–1) with clear cardinalities.**
- **Include associative entities wherever M:N relationships exist.**
- **Indicate foreign keys and participation constraints clearly**

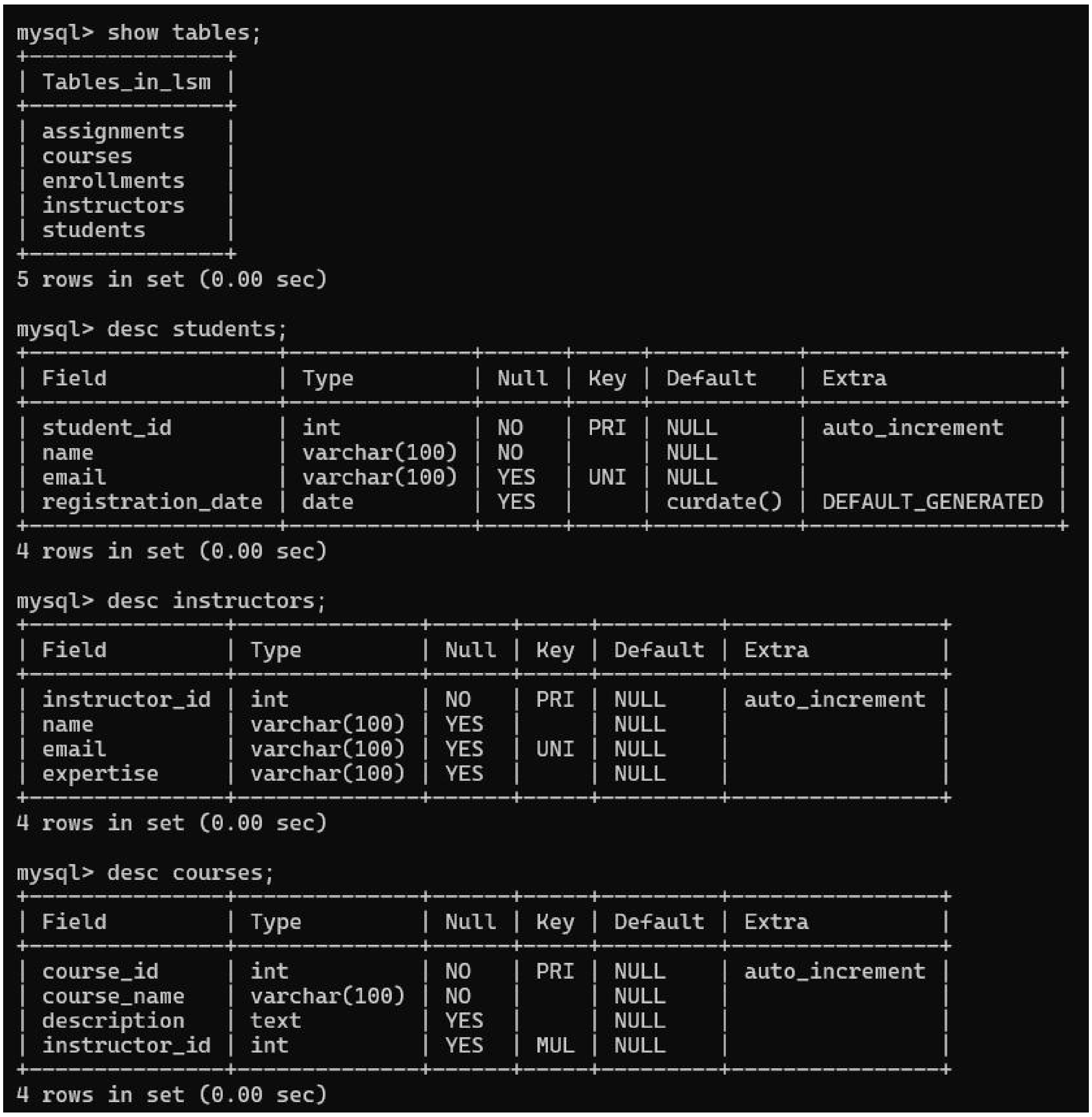


---

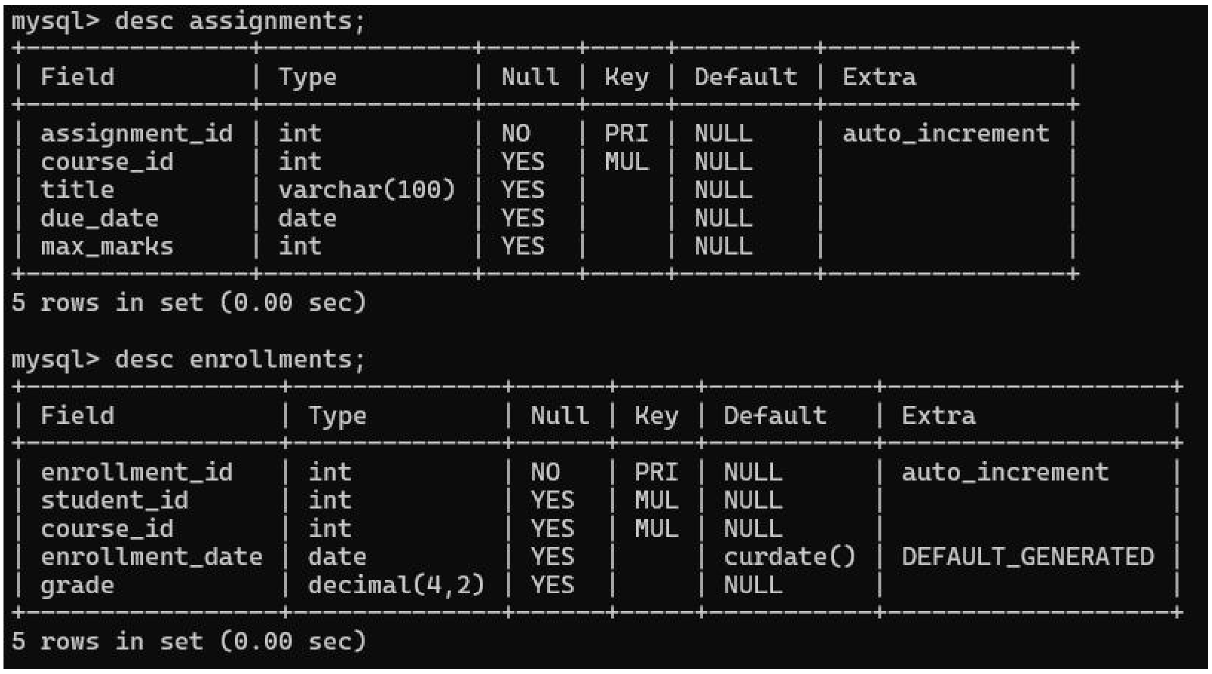
2) Create the database schema (DDL) with all required constraints and relationships. Appropriate data types and size definitions. Primary Keys and Foreign Keys for relationships. Unique, Check, and Not Null constraints. Use ENUM or SET data types where suitable (e.g., gender, status). Create indexes on key searchable fields.

```
CREATE DATABASE lsm;

USE lsm;

CREATE TABLE Students (

  student_id INT AUTO_INCREMENT PRIMARY KEY,
```

```sql
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE,
    registration_date DATE DEFAULT (CURRENT_DATE)
);

CREATE TABLE Instructors (
    instructor_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100) UNIQUE,
    expertise VARCHAR(100)
);

CREATE TABLE Courses (
    course_id INT AUTO_INCREMENT PRIMARY KEY,
    course_name VARCHAR(100) NOT NULL,
    description TEXT,
    instructor_id INT,
    FOREIGN KEY (instructor_id) REFERENCES Instructors(instructor_id)
);

CREATE TABLE Enrollments (
    enrollment_id INT AUTO_INCREMENT PRIMARY KEY,
    student_id INT,
    course_id INT,
    enrollment_date DATE DEFAULT (CURRENT_DATE),
    grade DECIMAL(4,2),
    FOREIGN KEY (student_id) REFERENCES Students(student_id),
    FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);

CREATE TABLE Assignments (
    assignment_id INT AUTO_INCREMENT PRIMARY KEY,
    course_id INT,
    title VARCHAR(100),
    due_date DATE,
    max_marks INT,
    FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);
```



```
mysql> show tables;
+----------------+
| Tables_in_lsm  |
+----------------+
| assignments    |
| courses        |
| enrollments    |
| instructors    |
| students       |
+----------------+
5 rows in set (0.00 sec)

mysql> desc students;
+-------------------+--------------+------+-----+-----------+-------------------+
| Field             | Type         | Null | Key | Default   | Extra             |
+-------------------+--------------+------+-----+-----------+-------------------+
| student_id        | int          | NO   | PRI | NULL      | auto_increment    |
| name              | varchar(100) | NO   |     | NULL      |                   |
| email             | varchar(100) | YES  | UNI | NULL      |                   |
| registration_date | date         | YES  |     | curdate() | DEFAULT_GENERATED |
+-------------------+--------------+------+-----+-----------+-------------------+
4 rows in set (0.00 sec)

mysql> desc instructors;
+---------------+--------------+------+-----+---------+----------------+
| Field         | Type         | Null | Key | Default | Extra          |
+---------------+--------------+------+-----+---------+----------------+
| instructor_id | int          | NO   | PRI | NULL    | auto_increment |
| name          | varchar(100) | YES  |     | NULL    |                |
| email         | varchar(100) | YES  | UNI | NULL    |                |
| expertise     | varchar(100) | YES  |     | NULL    |                |
+---------------+--------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)

mysql> desc courses;
+---------------+--------------+------+-----+---------+----------------+
| Field         | Type         | Null | Key | Default | Extra          |
+---------------+--------------+------+-----+---------+----------------+
| course_id     | int          | NO   | PRI | NULL    | auto_increment |
| course_name   | varchar(100) | NO   |     | NULL    |                |
| description   | text         | YES  |     | NULL    |                |
| instructor_id | int          | YES  | MUL | NULL    |                |
+---------------+--------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```



```
mysql> desc assignments;
+---------------+--------------+------+-----+---------+----------------+
| Field         | Type         | Null | Key | Default | Extra          |
+---------------+--------------+------+-----+---------+----------------+
| assignment_id | int          | NO   | PRI | NULL    | auto_increment |
| course_id     | int          | YES  | MUL | NULL    |                |
| title         | varchar(100) | YES  |     | NULL    |                |
| due_date      | date         | YES  |     | NULL    |                |
| max_marks     | int          | YES  |     | NULL    |                |
+---------------+--------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)

mysql> desc enrollments;
+-----------------+--------------+------+-----+-----------+-------------------+
| Field           | Type         | Null | Key | Default   | Extra             |
+-----------------+--------------+------+-----+-----------+-------------------+
| enrollment_id   | int          | NO   | PRI | NULL      | auto_increment    |
| student_id      | int          | YES  | MUL | NULL      |                   |
| course_id       | int          | YES  | MUL | NULL      |                   |
| enrollment_date | date         | YES  |     | curdate() | DEFAULT_GENERATED |
| grade           | decimal(4,2) | YES  |     | NULL      |                   |
+-----------------+--------------+------+-----+-----------+-------------------+
5 rows in set (0.00 sec)
```

----

3) Perform DML operations (Insert, Update, Delete) to populate sample data. Insert at least 5–10 records in each main table. Update some attribute (e.g., change contact info, modify price, update status). Delete one or more records safely (with WHERE condition).

INSERT INTO Instructors (name, email, expertise)

VALUES

('Dr. Kiran', 'kiran@lms.com', 'Data Science'),

('Prof. Aditya', 'aditya@lms.com', 'Web Development'),

('Dr. Vipul', 'vipul@lms.com', 'Database Systems'),

('Prof. Aniket', 'aniket@lms.com', 'Machine Learning'),

('Dr. Shilbhushan', 'shilbhushan@lms.com', 'Cloud Computing');

INSERT INTO Courses (course_name, description, instructor_id)

VALUES

('Python Basics', 'Introduction to Python Programming', 1),

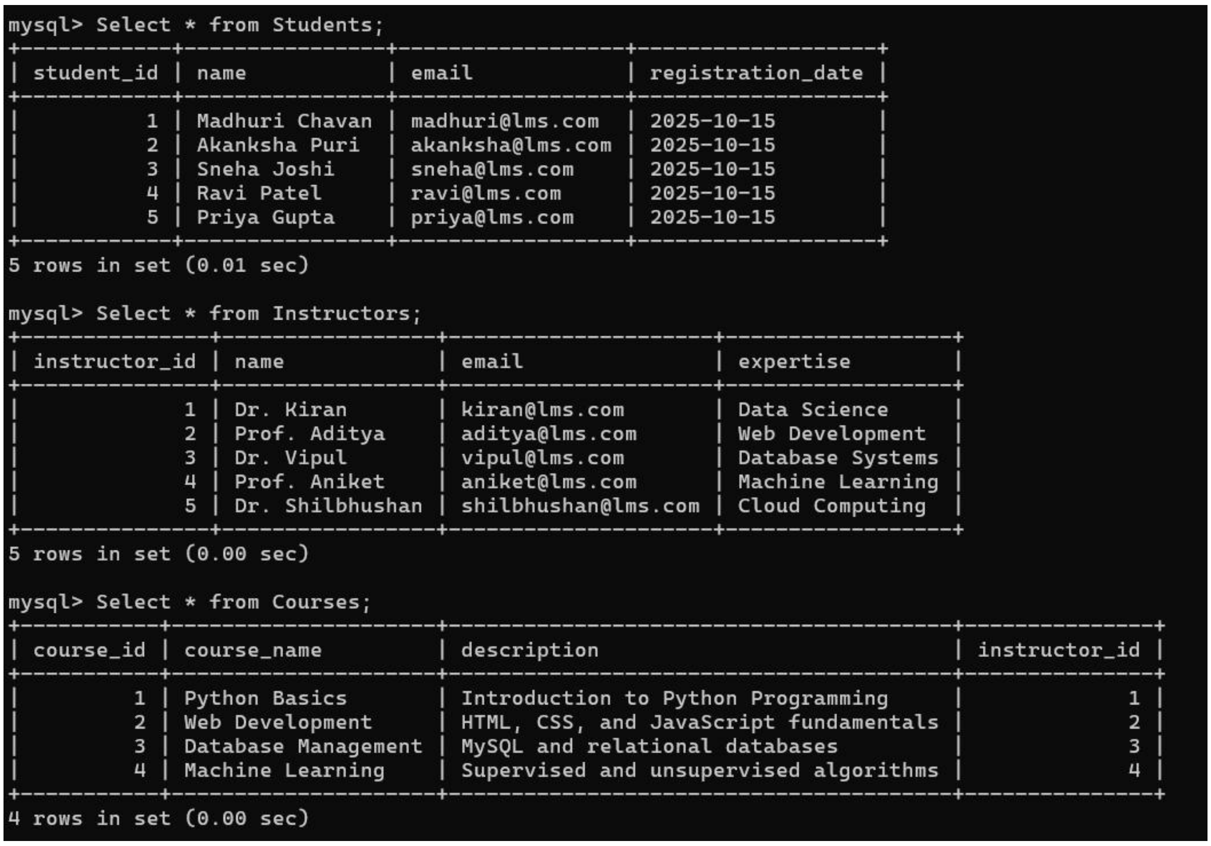
('Web Development', 'HTML, CSS, and JavaScript fundamentals', 2),

('Database Management', 'MySQL and relational databases', 3),

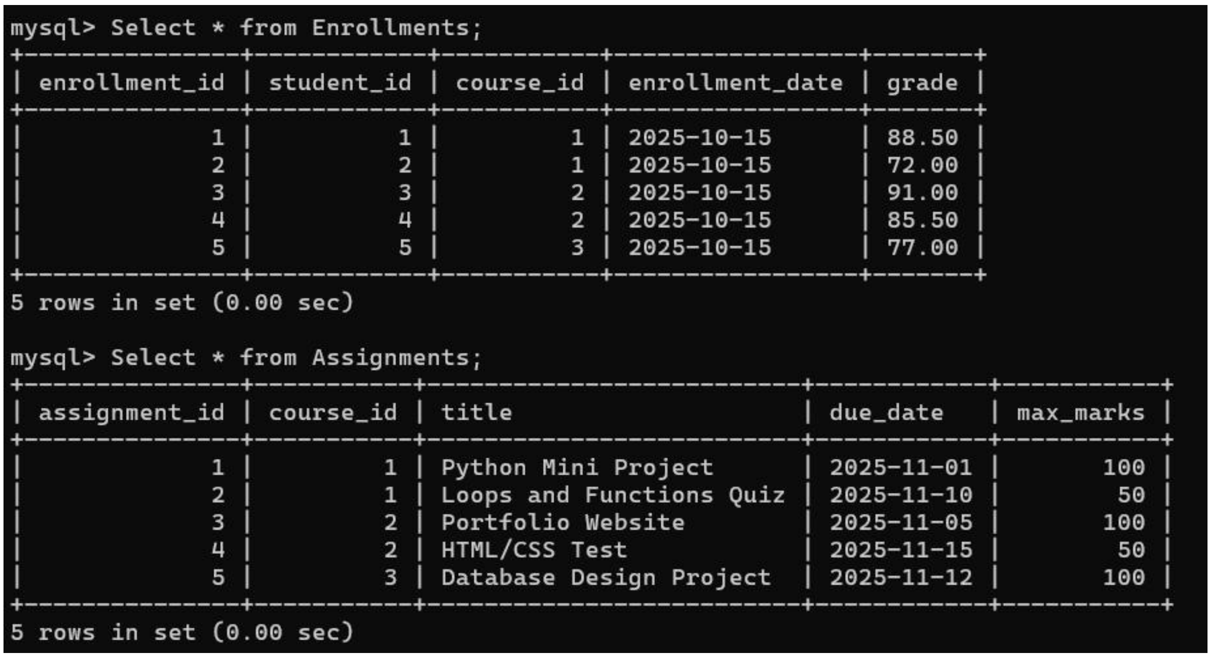
('Machine Learning', 'Supervised and unsupervised algorithms', 4);

INSERT INTO Students (name, email)

```sql
VALUES
('Madhuri Chavan', 'madhuri@lms.com'),
('Akanksha Puri', 'akanksha@lms.com'),
('Sneha Joshi', 'sneha@lms.com'),
('Ravi Patel', 'ravi@lms.com'),
('Priya Gupta', 'priya@lms.com');

-- Adjusted Enrollments to reference student_id 1-5 and course_id 1-4
INSERT INTO Enrollments (student_id, course_id, grade)
VALUES
(1, 1, 88.5),
(2, 1, 72.0),
(3, 2, 91.0),
(4, 2, 85.5),
(5, 3, 77.0);

INSERT INTO Assignments (course_id, title, due_date, max_marks)
VALUES
(1, 'Python Mini Project', '2025-11-01', 100),
(1, 'Loops and Functions Quiz', '2025-11-10', 50),
(2, 'Portfolio Website', '2025-11-05', 100),
(2, 'HTML/CSS Test', '2025-11-15', 50),
(3, 'Database Design Project', '2025-11-12', 100);
```



```
mysql> Select * from Students;
+------------+----------------+------------------+-------------------+
| student_id | name           | email            | registration_date |
+------------+----------------+------------------+-------------------+
|          1 | Madhuri Chavan | madhuri@lms.com  | 2025-10-15        |
|          2 | Akanksha Puri  | akanksha@lms.com | 2025-10-15        |
|          3 | Sneha Joshi    | sneha@lms.com    | 2025-10-15        |
|          4 | Ravi Patel     | ravi@lms.com     | 2025-10-15        |
|          5 | Priya Gupta    | priya@lms.com    | 2025-10-15        |
+------------+----------------+------------------+-------------------+
5 rows in set (0.01 sec)

mysql> Select * from Instructors;
+---------------+-----------------+---------------------+-------------------+
| instructor_id | name            | email               | expertise         |
+---------------+-----------------+---------------------+-------------------+
|             1 | Dr. Kiran       | kiran@lms.com       | Data Science      |
|             2 | Prof. Aditya    | aditya@lms.com      | Web Development   |
|             3 | Dr. Vipul       | vipul@lms.com       | Database Systems  |
|             4 | Prof. Aniket    | aniket@lms.com      | Machine Learning  |
|             5 | Dr. Shilbhushan | shilbhushan@lms.com | Cloud Computing   |
+---------------+-----------------+---------------------+-------------------+
5 rows in set (0.00 sec)

mysql> Select * from Courses;
+-----------+---------------------+-----------------------------------------+---------------+
| course_id | course_name         | description                             | instructor_id |
+-----------+---------------------+-----------------------------------------+---------------+
|         1 | Python Basics       | Introduction to Python Programming      |             1 |
|         2 | Web Development     | HTML, CSS, and JavaScript fundamentals  |             2 |
|         3 | Database Management | MySQL and relational databases          |             3 |
|         4 | Machine Learning    | Supervised and unsupervised algorithms  |             4 |
+-----------+---------------------+-----------------------------------------+---------------+
4 rows in set (0.00 sec)
```



```
mysql> Select * from Enrollments;
+---------------+------------+-----------+-----------------+-------+
| enrollment_id | student_id | course_id | enrollment_date | grade |
+---------------+------------+-----------+-----------------+-------+
|             1 |          1 |         1 | 2025-10-15      | 88.50 |
|             2 |          2 |         1 | 2025-10-15      | 72.00 |
|             3 |          3 |         2 | 2025-10-15      | 91.00 |
|             4 |          4 |         2 | 2025-10-15      | 85.50 |
|             5 |          5 |         3 | 2025-10-15      | 77.00 |
+---------------+------------+-----------+-----------------+-------+
5 rows in set (0.00 sec)

mysql> Select * from Assignments;
+---------------+-----------+---------------------------+------------+-----------+
| assignment_id | course_id | title                     | due_date   | max_marks |
+---------------+-----------+---------------------------+------------+-----------+
|             1 |         1 | Python Mini Project       | 2025-11-01 |       100 |
|             2 |         1 | Loops and Functions Quiz  | 2025-11-10 |        50 |
|             3 |         2 | Portfolio Website         | 2025-11-05 |       100 |
|             4 |         2 | HTML/CSS Test             | 2025-11-15 |        50 |
|             5 |         3 | Database Design Project   | 2025-11-12 |       100 |
+---------------+-----------+---------------------------+------------+-----------+
5 rows in set (0.00 sec)
```

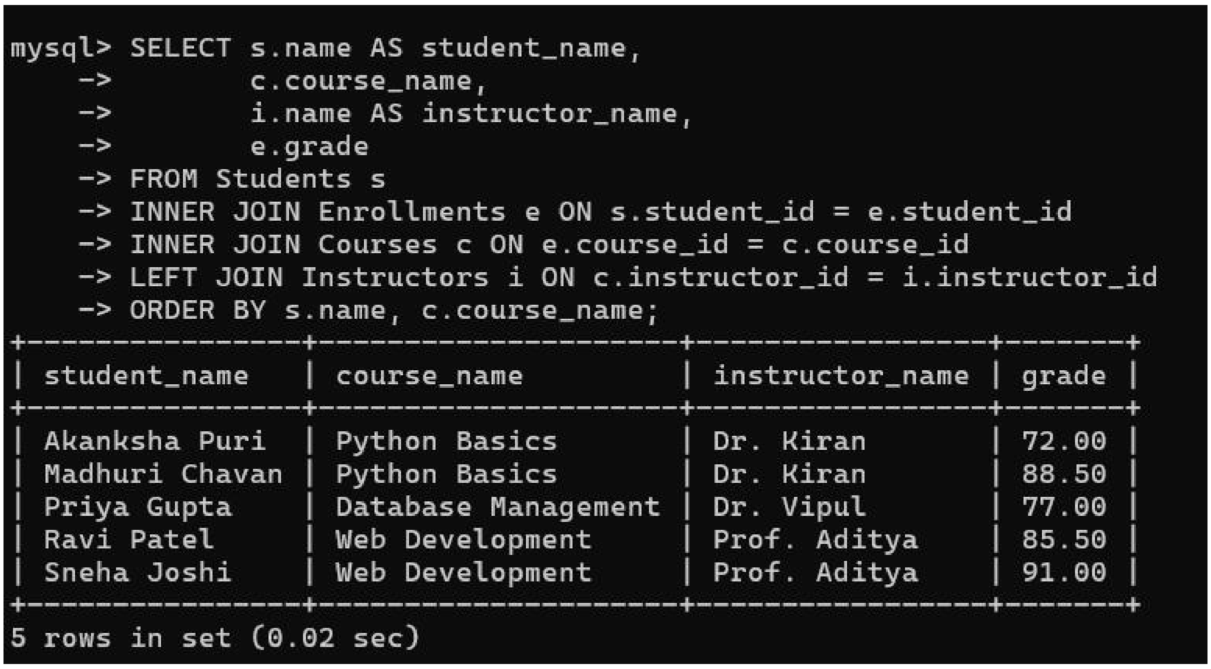
----

4) Write SQL Queries using Joins, Aggregate functions, Grouping, and Subqueries to retrieve meaningful information.

1: Join - List students, their enrolled courses, and instructor names

mysql> SELECT s.name AS student_name,

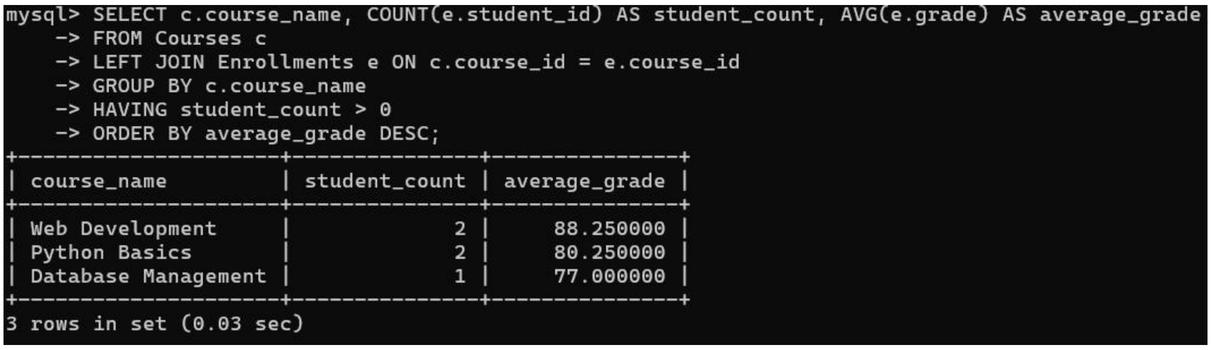
```
->        c.course_name,
->        i.name AS instructor_name,
->        e.grade
-> FROM Students s
-> INNER JOIN Enrollments e ON s.student_id = e.student_id
-> INNER JOIN Courses c ON e.course_id = c.course_id
-> LEFT JOIN Instructors i ON c.instructor_id = i.instructor_id
-> ORDER BY s.name, c.course_name;
```



```
mysql> SELECT s.name AS student_name,
    ->        c.course_name,
    ->        i.name AS instructor_name,
    ->        e.grade
    -> FROM Students s
    -> INNER JOIN Enrollments e ON s.student_id = e.student_id
    -> INNER JOIN Courses c ON e.course_id = c.course_id
    -> LEFT JOIN Instructors i ON c.instructor_id = i.instructor_id
    -> ORDER BY s.name, c.course_name;
+----------------+---------------------+-----------------+-------+
| student_name   | course_name         | instructor_name | grade |
+----------------+---------------------+-----------------+-------+
| Akanksha Puri  | Python Basics       | Dr. Kiran       | 72.00 |
| Madhuri Chavan | Python Basics       | Dr. Kiran       | 88.50 |
| Priya Gupta    | Database Management | Dr. Vipul       | 77.00 |
| Ravi Patel     | Web Development     | Prof. Aditya    | 85.50 |
| Sneha Joshi    | Web Development     | Prof. Aditya    | 91.00 |
+----------------+---------------------+-----------------+-------+
5 rows in set (0.02 sec)
```

2.Aggregate & Grouping - Average grade per course with student count

```
mysql> SELECT c.course_name, COUNT(e.student_id) AS student_count, AVG(e.grade) AS average_grade
-> FROM Courses c
-> LEFT JOIN Enrollments e ON c.course_id = e.course_id
-> GROUP BY c.course_name
-> HAVING student_count > 0
-> ORDER BY average_grade DESC;
```



```
mysql> SELECT c.course_name, COUNT(e.student_id) AS student_count, AVG(e.grade) AS average_grade
    -> FROM Courses c
    -> LEFT JOIN Enrollments e ON c.course_id = e.course_id
    -> GROUP BY c.course_name
    -> HAVING student_count > 0
    -> ORDER BY average_grade DESC;
+---------------------+---------------+---------------+
| course_name         | student_count | average_grade |
+---------------------+---------------+---------------+
| Web Development     |             2 |     88.250000 |
| Python Basics       |             2 |     80.250000 |
| Database Management |             1 |     77.000000 |
+---------------------+---------------+---------------+
3 rows in set (0.03 sec)
```

3. Subquery - Students with grades above the average grade of their course

SELECT s.name, e.grade, c.course_name

FROM Students s

INNER JOIN Enrollments e ON s.student_id = e.student_id

INNER JOIN Courses c ON e.course_id = c.course_id

WHERE e.grade > (

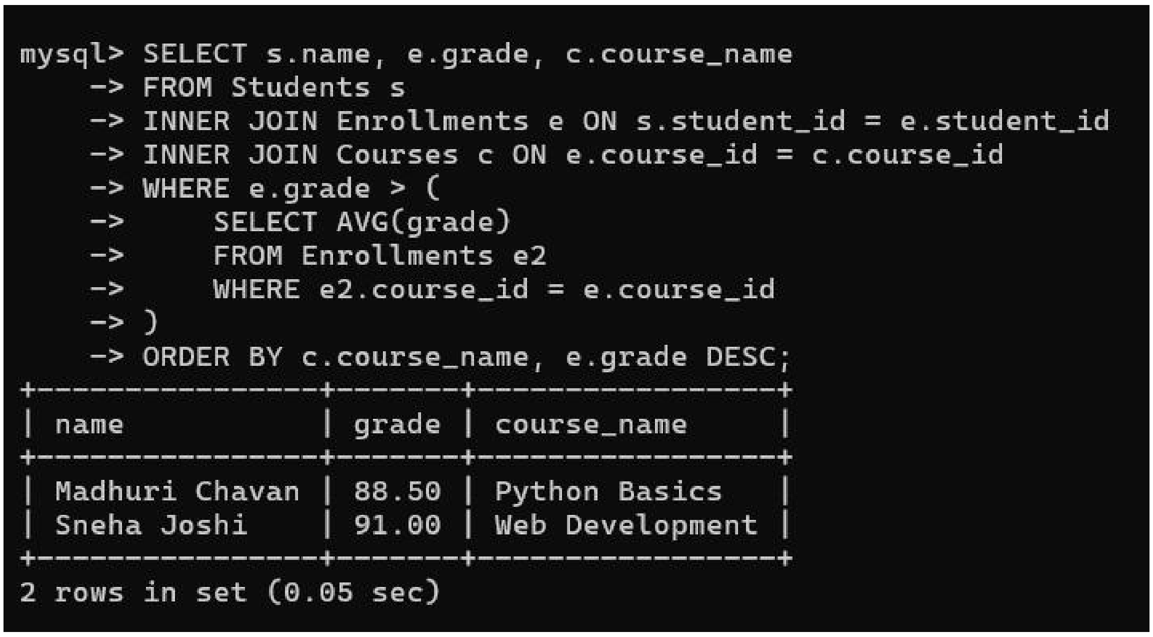
  SELECT AVG(grade)

  FROM Enrollments e2

  WHERE e2.course_id = e.course_id

)

ORDER BY c.course_name, e.grade DESC;



```
mysql> SELECT s.name, e.grade, c.course_name
    -> FROM Students s
    -> INNER JOIN Enrollments e ON s.student_id = e.student_id
    -> INNER JOIN Courses c ON e.course_id = c.course_id
    -> WHERE e.grade > (
    ->     SELECT AVG(grade)
    ->     FROM Enrollments e2
    ->     WHERE e2.course_id = e.course_id
    -> )
    -> ORDER BY c.course_name, e.grade DESC;
+----------------+-------+-----------------+
| name           | grade | course_name     |
+----------------+-------+-----------------+
| Madhuri Chavan | 88.50 | Python Basics   |
| Sneha Joshi    | 91.00 | Web Development |
+----------------+-------+-----------------+
2 rows in set (0.05 sec)
```

5: Subquery & Join - Students enrolled in courses taught by a specific instructor

SELECT s.name, s.email, c.course_name

FROM Students s

INNER JOIN Enrollments e ON s.student_id = e.student_id

INNER JOIN Courses c ON e.course_id = c.course_id

WHERE c.instructor_id = (

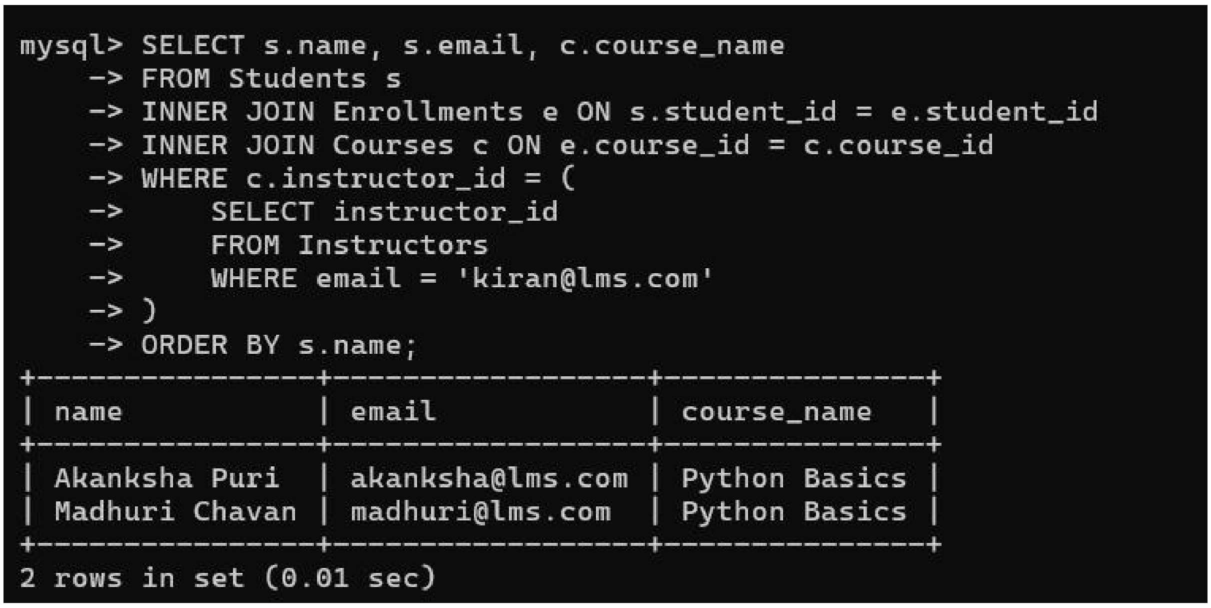
SELECT instructor_id

FROM Instructors

WHERE email = 'kiran@lms.com'

)

ORDER BY s.name;



```
mysql> SELECT s.name, s.email, c.course_name
    -> FROM Students s
    -> INNER JOIN Enrollments e ON s.student_id = e.student_id
    -> INNER JOIN Courses c ON e.course_id = c.course_id
    -> WHERE c.instructor_id = (
    ->     SELECT instructor_id
    ->     FROM Instructors
    ->     WHERE email = 'kiran@lms.com'
    -> )
    -> ORDER BY s.name;
+----------------+------------------+---------------+
| name           | email            | course_name   |
+----------------+------------------+---------------+
| Akanksha Puri  | akanksha@lms.com | Python Basics |
| Madhuri Chavan | madhuri@lms.com  | Python Basics |
+----------------+------------------+---------------+
2 rows in set (0.01 sec)
```

6: Aggregate, Grouping & Join - Gender-based enrollment statistics per course

SELECT c.course_name, s.gender, COUNT(e.student_id) AS enrollment_count

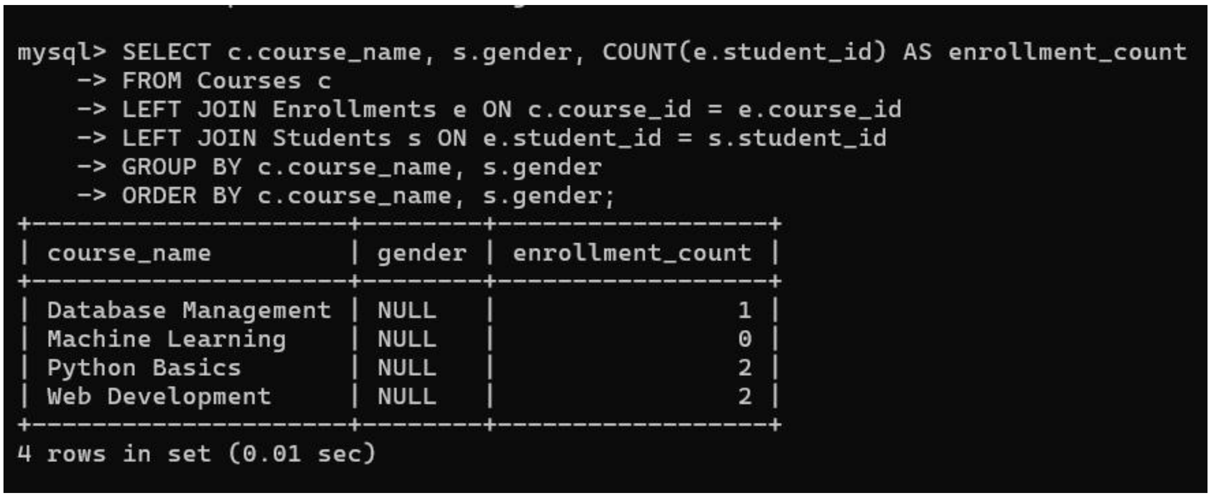
FROM Courses c

LEFT JOIN Enrollments e ON c.course_id = e.course_id

LEFT JOIN Students s ON e.student_id = s.student_id

GROUP BY c.course_name, s.gender

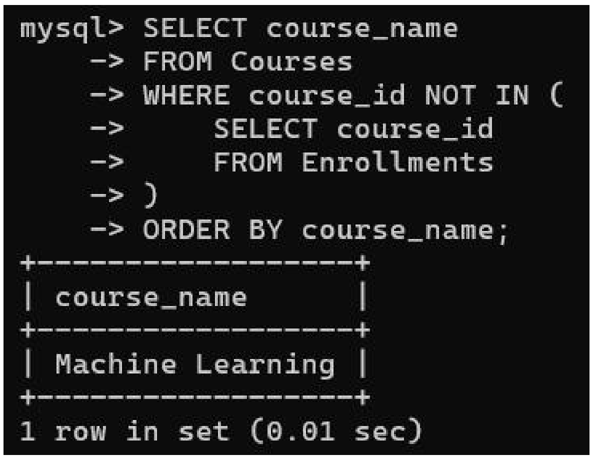
ORDER BY c.course_name, s.gender;



```
mysql> SELECT c.course_name, s.gender, COUNT(e.student_id) AS enrollment_count
    -> FROM Courses c
    -> LEFT JOIN Enrollments e ON c.course_id = e.course_id
    -> LEFT JOIN Students s ON e.student_id = s.student_id
    -> GROUP BY c.course_name, s.gender
    -> ORDER BY c.course_name, s.gender;
+---------------------+--------+------------------+
| course_name         | gender | enrollment_count |
+---------------------+--------+------------------+
| Database Management | NULL   |                1 |
| Machine Learning    | NULL   |                0 |
| Python Basics       | NULL   |                2 |
| Web Development     | NULL   |                2 |
+---------------------+--------+------------------+
4 rows in set (0.01 sec)
```

7: Subquery - Courses with no enrollments

```
SELECT course_name

FROM Courses

WHERE course_id NOT IN (

   SELECT course_id

   FROM Enrollments

)

ORDER BY course_name;
```



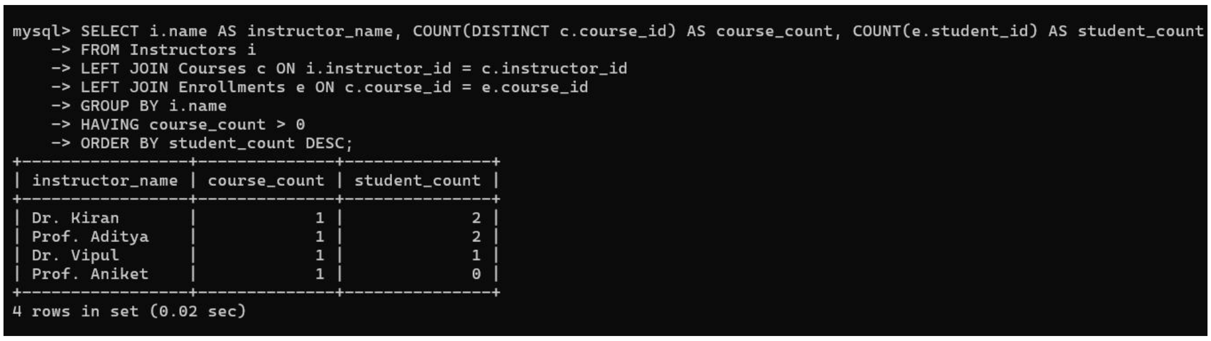
```
mysql> SELECT course_name
    -> FROM Courses
    -> WHERE course_id NOT IN (
    ->     SELECT course_id
    ->     FROM Enrollments
    -> )
    -> ORDER BY course_name;
+------------------+
| course_name      |
+------------------+
| Machine Learning |
+------------------+
1 row in set (0.01 sec)
```

8: Join, Aggregate & Grouping - Instructor teaching load (number of courses and students)

```
SELECT i.name AS instructor_name, COUNT(DISTINCT c.course_id) AS course_count,
COUNT(e.student_id) AS student_count

FROM Instructors i

LEFT JOIN Courses c ON i.instructor_id = c.instructor_id

LEFT JOIN Enrollments e ON c.course_id = e.course_id

GROUP BY i.name
```

HAVING course_count > 0

ORDER BY student_count DESC;



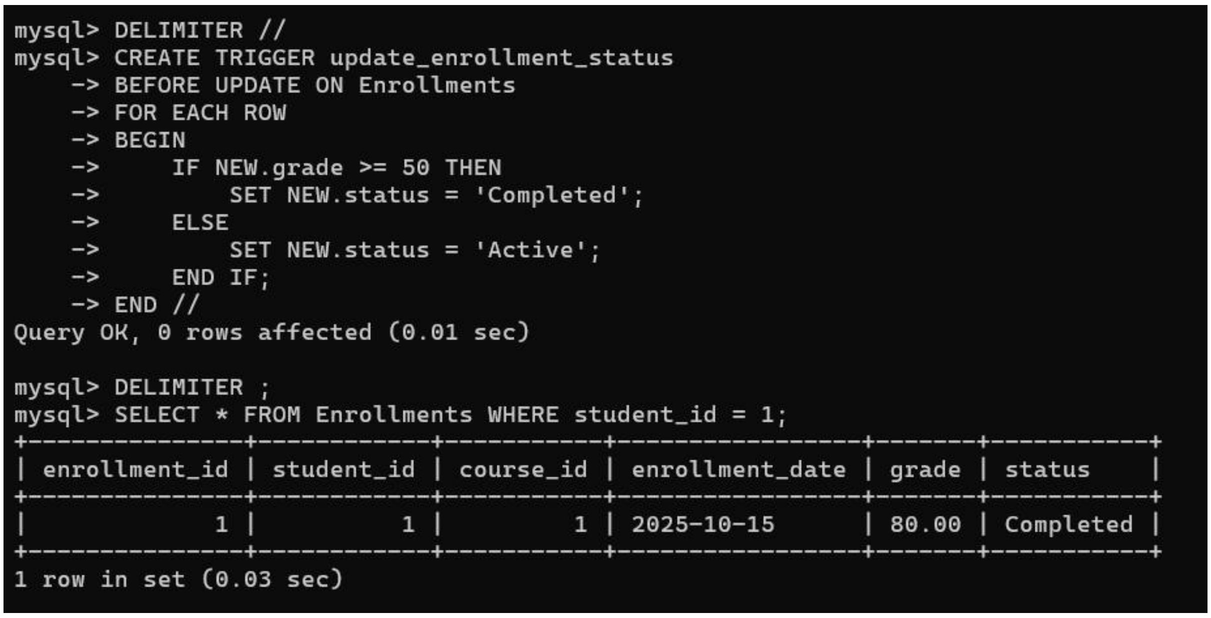
```
mysql> SELECT i.name AS instructor_name, COUNT(DISTINCT c.course_id) AS course_count, COUNT(e.student_id) AS student_count
    -> FROM Instructors i
    -> LEFT JOIN Courses c ON i.instructor_id = c.instructor_id
    -> LEFT JOIN Enrollments e ON c.course_id = e.course_id
    -> GROUP BY i.name
    -> HAVING course_count > 0
    -> ORDER BY student_count DESC;
+-----------------+--------------+---------------+
| instructor_name | course_count | student_count |
+-----------------+--------------+---------------+
| Dr. Kiran       |            1 |             2 |
| Prof. Aditya    |            1 |             2 |
| Dr. Vipul       |            1 |             1 |
| Prof. Aniket    |            1 |             0 |
+-----------------+--------------+---------------+
4 rows in set (0.02 sec)
```

----

5) Implement a Trigger, a Function, and a Stored Procedure relevant to your system's logic.

Trigger: Automatically update enrollment status based on grade

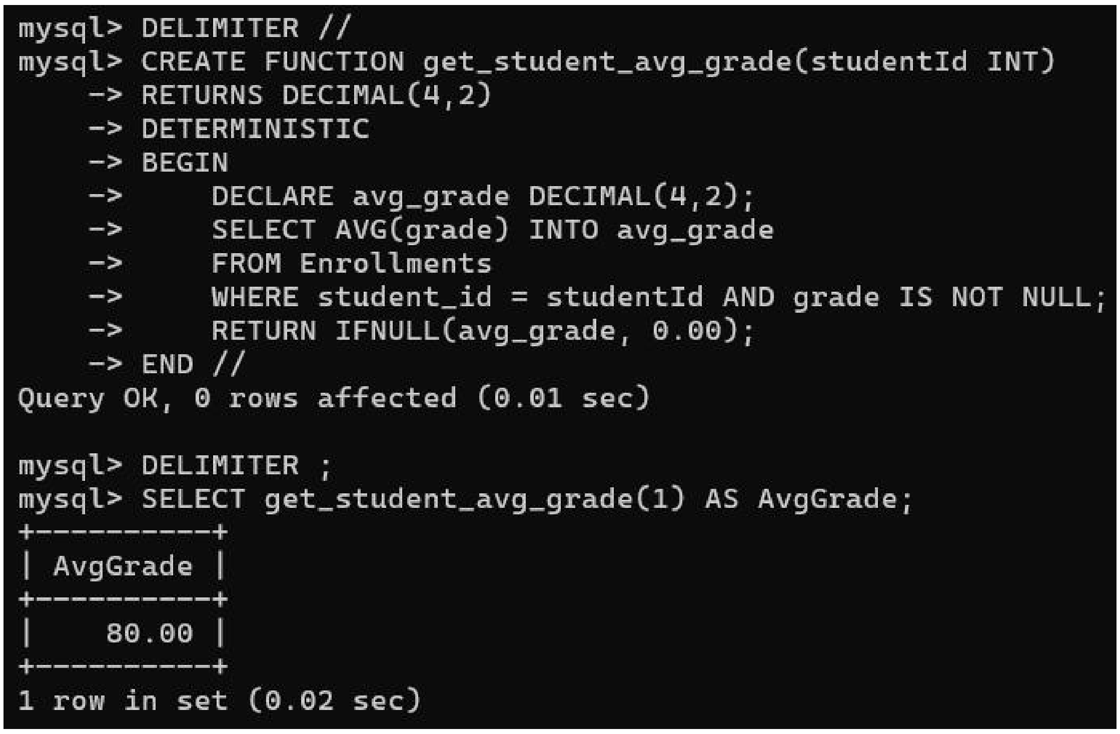
A trigger is a special stored procedure in a database that automatically executes when specific events (like INSERT, UPDATE, or DELETE) occur on a table. Triggers help automate tasks, maintain data consistency, and record database activities. Each trigger is tied to a particular table and runs without manual execution.



```
mysql> DELIMITER //
mysql> CREATE TRIGGER update_enrollment_status
    -> BEFORE UPDATE ON Enrollments
    -> FOR EACH ROW
    -> BEGIN
    ->     IF NEW.grade >= 50 THEN
    ->         SET NEW.status = 'Completed';
    ->     ELSE
    ->         SET NEW.status = 'Active';
    ->     END IF;
    -> END //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> SELECT * FROM Enrollments WHERE student_id = 1;
+---------------+------------+-----------+-----------------+-------+-----------+
| enrollment_id | student_id | course_id | enrollment_date | grade | status    |
+---------------+------------+-----------+-----------------+-------+-----------+
|             1 |          1 |         1 | 2025-10-15      | 80.00 | Completed |
+---------------+------------+-----------+-----------------+-------+-----------+
1 row in set (0.03 sec)
```

Function: Calculate a student's average grade across all enrolled courses

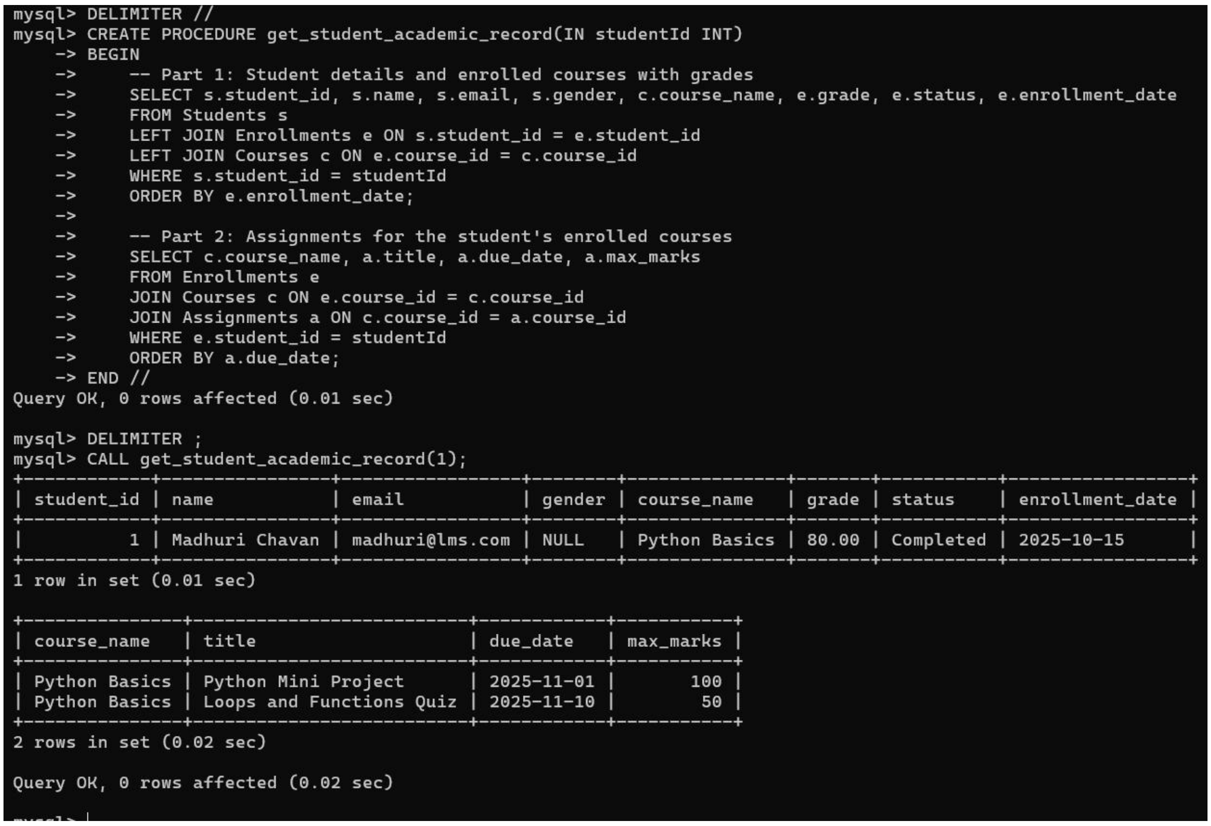
A function in SQL is a named block of code that performs a specific task and returns a single value. Functions are reusable and can be called within queries, expressions, or other functions.



```
mysql> DELIMITER //
mysql> CREATE FUNCTION get_student_avg_grade(studentId INT)
    -> RETURNS DECIMAL(4,2)
    -> DETERMINISTIC
    -> BEGIN
    ->     DECLARE avg_grade DECIMAL(4,2);
    ->     SELECT AVG(grade) INTO avg_grade
    ->     FROM Enrollments
    ->     WHERE student_id = studentId AND grade IS NOT NULL;
    ->     RETURN IFNULL(avg_grade, 0.00);
    -> END //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> SELECT get_student_avg_grade(1) AS AvgGrade;
+----------+
| AvgGrade |
+----------+
|    80.00 |
+----------+
1 row in set (0.02 sec)
```

Stored Procedure: Retrieve a student's academic record (courses and assignments)

A Stored Procedure in SQL is a named set of SQL statements that you can save and execute as a unit. Unlike a function, it may or may not return a value, and it can perform multiple operations, including modifying tables.



```
mysql> DELIMITER //
mysql> CREATE PROCEDURE get_student_academic_record(IN studentId INT)
    -> BEGIN
    ->     -- Part 1: Student details and enrolled courses with grades
    ->     SELECT s.student_id, s.name, s.email, s.gender, c.course_name, e.grade, e.status, e.enrollment_date
    ->     FROM Students s
    ->     LEFT JOIN Enrollments e ON s.student_id = e.student_id
    ->     LEFT JOIN Courses c ON e.course_id = c.course_id
    ->     WHERE s.student_id = studentId
    ->     ORDER BY e.enrollment_date;
    ->
    ->     -- Part 2: Assignments for the student's enrolled courses
    ->     SELECT c.course_name, a.title, a.due_date, a.max_marks
    ->     FROM Enrollments e
    ->     JOIN Courses c ON e.course_id = c.course_id
    ->     JOIN Assignments a ON c.course_id = a.course_id
    ->     WHERE e.student_id = studentId
    ->     ORDER BY a.due_date;
    -> END //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> CALL get_student_academic_record(1);
+------------+----------------+------------------+--------+---------------+-------+-----------+-----------------+
| student_id | name           | email            | gender | course_name   | grade | status    | enrollment_date |
+------------+----------------+------------------+--------+---------------+-------+-----------+-----------------+
|          1 | Madhuri Chavan | madhuri@lms.com  | NULL   | Python Basics | 80.00 | Completed | 2025-10-15      |
+------------+----------------+------------------+--------+---------------+-------+-----------+-----------------+
1 row in set (0.01 sec)

+---------------+--------------------------+------------+-----------+
| course_name   | title                    | due_date   | max_marks |
+---------------+--------------------------+------------+-----------+
| Python Basics | Python Mini Project      | 2025-11-01 |       100 |
| Python Basics | Loops and Functions Quiz | 2025-11-10 |        50 |
+---------------+--------------------------+------------+-----------+
2 rows in set (0.02 sec)

Query OK, 0 rows affected (0.02 sec)
```

---

6) Normalize your database up to Third Normal Form (3NF) and provide a short explanation.

- **Identify repeating groups → convert to 1NF.**
- **Remove partial dependencies → convert to 2NF.**
- **Remove transitive dependencies → convert to 3NF.**
- **Clearly show the final normalized tables.**
- **Explain each step briefly**

<u>Step-by-Step Normalization</u>

Step 1: First Normal Form (1NF)

Goal: Eliminate repeating groups and ensure all attributes are atomic, with each table having a primary key.

Analysis:

- **Current Schema**: The schema is already in 1NF because:
    - No repeating groups exist (e.g., no multi-valued attributes like multiple emails in a single column).

Step 2: Second Normal Form (2NF)

Goal: Ensure the schema is in 1NF and remove partial dependencies (non-key attributes dependent on only part of a composite primary key).

Analysis:

- **Primary Keys**: Most tables (Students, Instructors, Courses, Assignments) have single-column primary keys (student_id, instructor_id, course_id, assignment_id), so partial dependencies are not possible.

Step 3: Third Normal Form (3NF)

Goal: Ensure the schema is in 2NF and remove transitive dependencies (non-key attributes dependent on other non-key attributes).

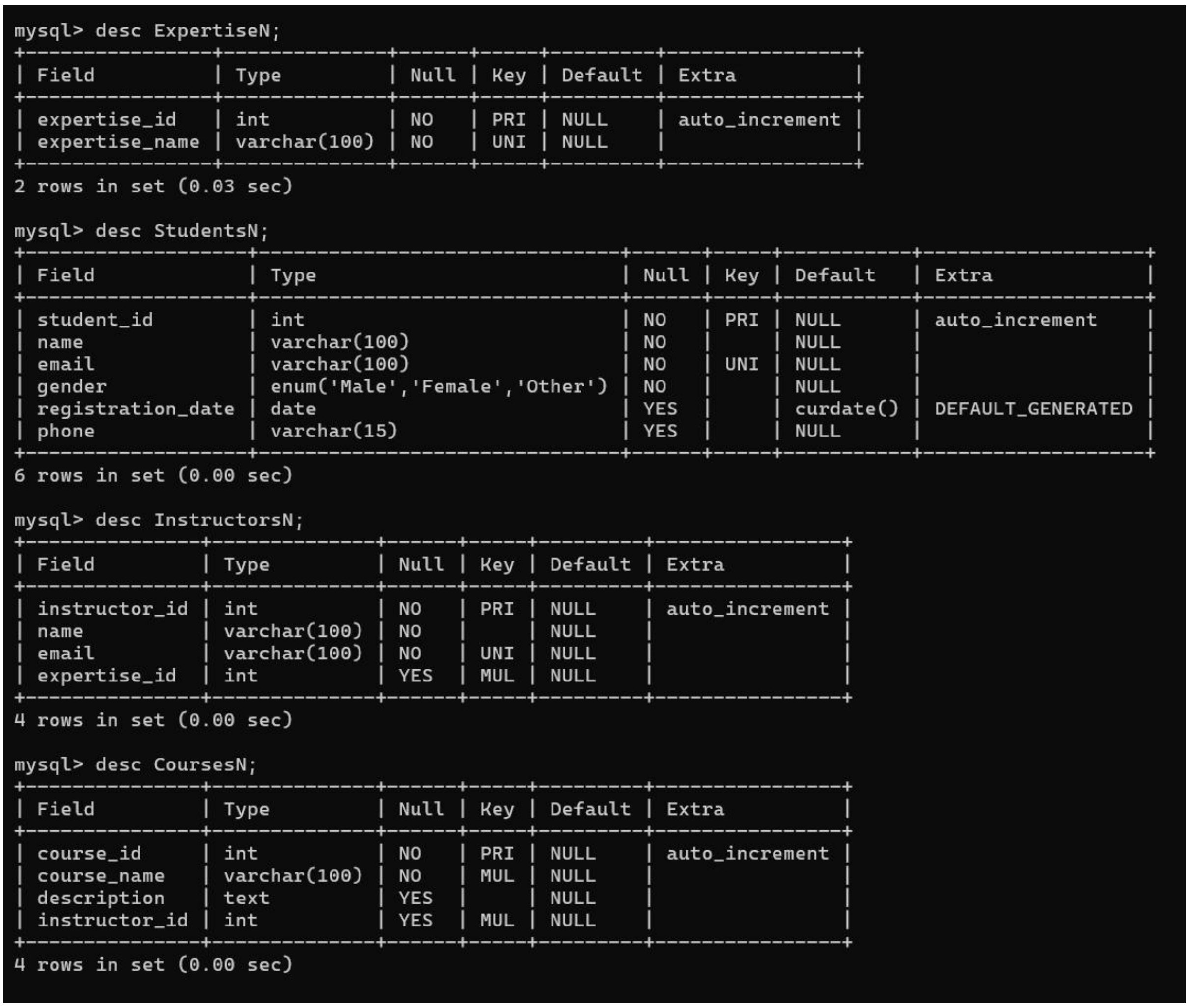
Analysis:

- **Transitive Dependencies**: A transitive dependency occurs when a non-key attribute depends on another non-key attribute, which in turn depends on the primary key.

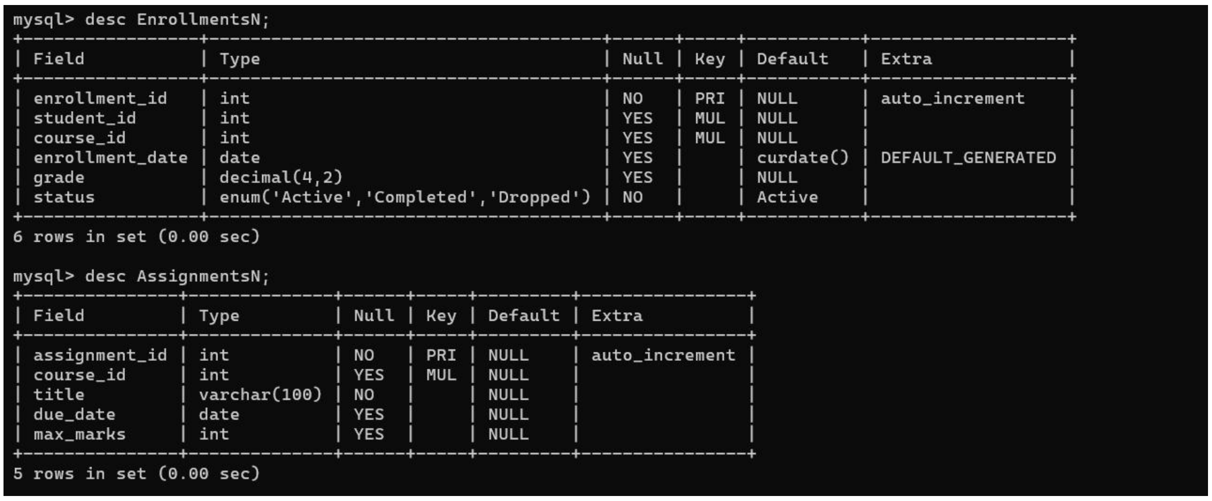
<u>Clearly show the final normalized tables.</u>

```
-- Creating Expertise table to normalize instructor expertise

CREATE TABLE ExpertiseN (

  expertise_id INT AUTO_INCREMENT PRIMARY KEY,

  expertise_name VARCHAR(100) NOT NULL UNIQUE

);
```

```
-- Creating Students table
CREATE TABLE StudentsN (
   student_id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(100) NOT NULL,
   email VARCHAR(100) UNIQUE NOT NULL,
   gender ENUM('Male', 'Female', 'Other') NOT NULL,
   registration_date DATE DEFAULT (CURRENT_DATE),
   phone VARCHAR(15),
   INDEX idx_student_email (email)
);

-- Creating Instructors table with foreign key to Expertise
CREATE TABLE InstructorsN (
   instructor_id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(100) NOT NULL,
   email VARCHAR(100) UNIQUE NOT NULL,
   expertise_id INT,
   FOREIGN KEY (expertise_id) REFERENCES Expertise(expertise_id) ON DELETE SET NULL ON
UPDATE CASCADE,
   INDEX idx_instructor_email (email)
);

-- Creating Courses table
CREATE TABLE CoursesN (
   course_id INT AUTO_INCREMENT PRIMARY KEY,
   course_name VARCHAR(100) NOT NULL,
   description TEXT,
   instructor_id INT,
   FOREIGN KEY (instructor_id) REFERENCES Instructors(instructor_id) ON DELETE SET NULL
ON UPDATE CASCADE,
    INDEX idx_course_name (course_name)
);

-- Creating Enrollments table
CREATE TABLE EnrollmentsN (
    enrollment_id INT AUTO_INCREMENT PRIMARY KEY,
    student_id INT,
    course_id INT,
    enrollment_date DATE DEFAULT (CURRENT_DATE),
    grade DECIMAL(4,2) CHECK (grade >= 0 AND grade <= 100),
    status ENUM('Active', 'Completed', 'Dropped') NOT NULL DEFAULT 'Active',
    FOREIGN KEY (student_id) REFERENCES Students(student_id) ON DELETE CASCADE ON
UPDATE CASCADE,
    FOREIGN KEY (course_id) REFERENCES Courses(course_id) ON DELETE CASCADE ON
UPDATE CASCADE
);

-- Creating Assignments table
CREATE TABLE AssignmentsN (
    assignment_id INT AUTO_INCREMENT PRIMARY KEY,
    course_id INT,
    title VARCHAR(100) NOT NULL,
    due_date DATE,
    max_marks INT CHECK (max_marks > 0),
    FOREIGN KEY (course_id) REFERENCES Courses(course_id) ON DELETE CASCADE ON
UPDATE CASCADE
);
```



```
mysql> desc ExpertiseN;
+----------------+--------------+------+-----+---------+----------------+
| Field          | Type         | Null | Key | Default | Extra          |
+----------------+--------------+------+-----+---------+----------------+
| expertise_id   | int          | NO   | PRI | NULL    | auto_increment |
| expertise_name | varchar(100) | NO   | UNI | NULL    |                |
+----------------+--------------+------+-----+---------+----------------+
2 rows in set (0.03 sec)

mysql> desc StudentsN;
+-------------------+--------------------------------+------+-----+-----------+-------------------+
| Field             | Type                           | Null | Key | Default   | Extra             |
+-------------------+--------------------------------+------+-----+-----------+-------------------+
| student_id        | int                            | NO   | PRI | NULL      | auto_increment    |
| name              | varchar(100)                   | NO   |     | NULL      |                   |
| email             | varchar(100)                   | NO   | UNI | NULL      |                   |
| gender            | enum('Male','Female','Other')  | NO   |     | NULL      |                   |
| registration_date | date                           | YES  |     | curdate() | DEFAULT_GENERATED |
| phone             | varchar(15)                    | YES  |     | NULL      |                   |
+-------------------+--------------------------------+------+-----+-----------+-------------------+
6 rows in set (0.00 sec)

mysql> desc InstructorsN;
+---------------+--------------+------+-----+---------+----------------+
| Field         | Type         | Null | Key | Default | Extra          |
+---------------+--------------+------+-----+---------+----------------+
| instructor_id | int          | NO   | PRI | NULL    | auto_increment |
| name          | varchar(100) | NO   |     | NULL    |                |
| email         | varchar(100) | NO   | UNI | NULL    |                |
| expertise_id  | int          | YES  | MUL | NULL    |                |
+---------------+--------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)

mysql> desc CoursesN;
+---------------+--------------+------+-----+---------+----------------+
| Field         | Type         | Null | Key | Default | Extra          |
+---------------+--------------+------+-----+---------+----------------+
| course_id     | int          | NO   | PRI | NULL    | auto_increment |
| course_name   | varchar(100) | NO   | MUL | NULL    |                |
| description   | text         | YES  |     | NULL    |                |
| instructor_id | int          | YES  | MUL | NULL    |                |
+---------------+--------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```



```
mysql> desc EnrollmentsN;
+-----------------+--------------------------------------+------+-----+-----------+-------------------+
| Field           | Type                                 | Null | Key | Default   | Extra             |
+-----------------+--------------------------------------+------+-----+-----------+-------------------+
| enrollment_id   | int                                  | NO   | PRI | NULL      | auto_increment    |
| student_id      | int                                  | YES  | MUL | NULL      |                   |
| course_id       | int                                  | YES  | MUL | NULL      |                   |
| enrollment_date | date                                 | YES  |     | curdate() | DEFAULT_GENERATED |
| grade           | decimal(4,2)                         | YES  |     | NULL      |                   |
| status          | enum('Active','Completed','Dropped') | NO   |     | Active    |                   |
+-----------------+--------------------------------------+------+-----+-----------+-------------------+
6 rows in set (0.00 sec)

mysql> desc AssignmentsN;
+---------------+--------------+------+-----+---------+----------------+
| Field         | Type         | Null | Key | Default | Extra          |
+---------------+--------------+------+-----+---------+----------------+
| assignment_id | int          | NO   | PRI | NULL    | auto_increment |
| course_id     | int          | YES  | MUL | NULL    |                |
| title         | varchar(100) | NO   |     | NULL    |                |
| due_date      | date         | YES  |     | NULL    |                |
| max_marks     | int          | YES  |     | NULL    |                |
+---------------+--------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

1NF (Eliminate Repeating Groups):

- **Check**: Ensured no repeating groups or multi-valued attributes exist. All tables have primary keys, and attributes are atomic.
- **Outcome**: Schema was already in 1NF (no changes needed).

2NF (Remove Partial Dependencies):

- **Check**: Verified that no non-key attributes depend on part of a composite primary key. All tables have single-column primary keys, so partial dependencies are not possible.
- **Outcome**: Schema was already in 2NF (no changes needed).

3NF (Remove Transitive Dependencies):

- **Check**: Analyzed for non-key attributes depending on other non-key attributes. No clear transitive dependencies were found in the original schema, but expertise in Instructors was identified as a candidate for normalization to prevent potential redundancy.
- **Action**: Created an Expertise table with expertise_id and expertise_name, and modified Instructors to reference expertise_id as a foreign key. This ensures that expertise values (e.g., 'Data Science') are stored once, reducing redundancy and update anomalies.
- **Outcome**: Schema is now in 3NF, with the addition of the Expertise table to handle potential transitive dependencies.