DATABASE MANAGEMENT SYSTEM
MINI-PROJECT REPORT

NAME 1: SHIVAM SONI
SRN 1: PES1UG23AM287

NAME 2: SAIRAM S
SRN 2: PES1UG23AM257

Problem Abstract

This document outlines the specifications for a comprehensive 'Online Course Registration System', modeled after a university's student information system.
The system is architected around a robust MySQL database backend, which houses all data and encapsulates critical business logic through stored procedures, triggers, and functions.
The primary goal of this project is to develop a normalized database for handling the course registration process and an intuitive Graphical User Interface (GUI) for both students and administrators.
This GUI serves as a dynamic management dashboard, replacing the need for direct manual SQL commands and streamlining all administrative and student-led tasks, including viewing course catalogs, managing registrations, updating grades, and enforcing academic rules like prerequisites and classroom capacity.

User Requirement Specifications:

- Functional Requirements:

User Role Management:

The system shall support two primary user roles: Student and Admin/Instructor.

The GUI shall provide distinct, tabbed views tailored to the functionalities of each role.

The system shall allow users to dynamically select their profile from a drop-down list of all available students or instructors in the database

Student Dashboard & Registration:

The system shall display a personalized dashboard for the selected student, showing a complete list of their current registrations.

The system shall display a list of all available courses for registration from the student's department.

The system must allow a student to register for an available course. Registration shall be blocked if:
The course is already full (enforced by the GetAvailableSeats function).

The student has not completed the prerequisite course (enforced by
the HasCompletedPrerequisite function).

The system must allow a student to drop an existing registration, provided no payment has been made for it.

Admin/Instructor Dashboard & Management:

The system shall display a list of student registrations for courses taught only by the selected instructor.

The administrator must be able to update the grade for any student registration in their managed courses by invoking a stored procedure.

Automated Rule Enforcement & Auditing:

Upon a grade being updated, the system must automatically create a record in
the GradeChangeAudit table, logging the old

grade, the new grade, and a timestamp. This is enforced by a database trigger.

The system must prevent registrations in full classes and for courses where prerequisites are not met, enforced by database functions.

The system must maintain referential integrity using foreign key constraints (e.g., a student cannot be deleted if they have existing registrations).

- Non-Functional Requirements

Usability: The GUI must be intuitive and easy to navigate for non-technical users. The tabbed interface provides a clear separation of concerns for Student and Admin views.

Performance: The application must feel responsive. Database queries are optimized, and the frontend dynamically loads data without page reloads.

Data Integrity: All operations must maintain the consistency of the database. The system prevents invalid states, such as registering a student for a non-existent course.

Security: Database credentials used by the application are stored securely as environment variables on the server and are not exposed in the client-side code.

- Interface Requirements

The main application window features a tabbed layout:

1. Tab 1 (Student View):

Dashboard: A summary view showing the student's current registrations and all available courses for enrollment.

Manage Registration: Interface for dropping a course and for checking prerequisites using the SQL function.

2. Tab 2 (Admin/Instructor View):

Grade Management: An interface for selecting a student registration and updating their grade, which invokes the UpdateStudentGrade stored procedure.

Analytical Queries: Buttons to run complex pre-defined JOIN, nested, and aggregate queries to demonstrate advanced SQL capabilities.

Audit Log: A read-only view of the GradeChangeAudit table, showing all historical grade changes as captured by the database trigger.

List of Softwares, Tools, and Programming Languages

Database Management System: MySQL Server 8.0

Database Administration Tool: MySQL Workbench

Backend and Database Logic:
SQL - for defining the schema, stored procedures, triggers, and functions.

Backend Language: Python 3.x

Web Framework: Flask

Database Connector: mysql-connector-python

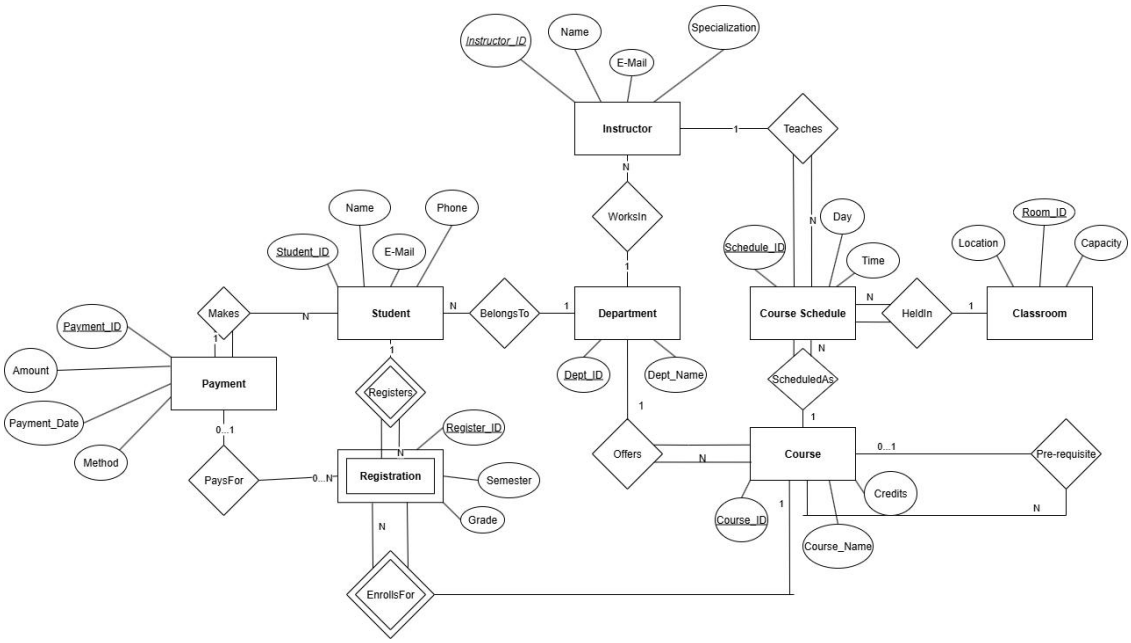
GUI Application:

Frontend Technology: HTML5, CSS3

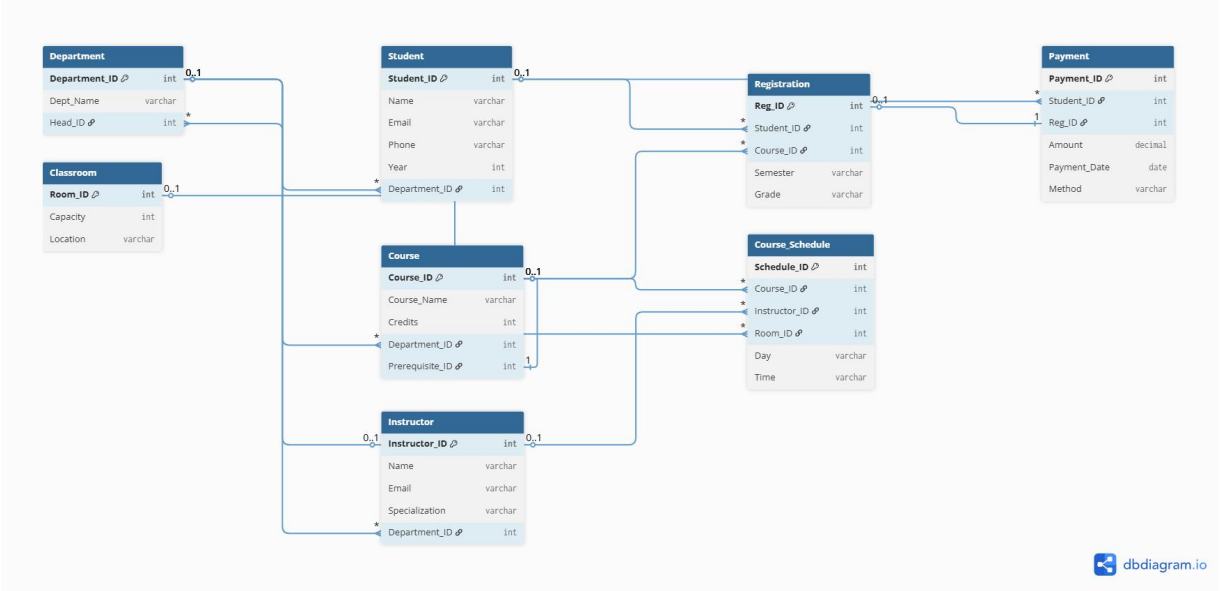
UI Framework: Tailwind CSS

Core Logic: Vanilla JavaScript (for API calls and DOM manipulation)

ER DIAGRAM:



RELATIONAL SCHEMA



DDL COMMANDS:

```
CREATE DATABASE IF NOT EXISTS online_registration_db_3;
USE online_registration_db_3;

CREATE TABLE Department (
    Department_ID INT PRIMARY KEY,
    Dept_Name VARCHAR(100) NOT NULL UNIQUE,
    Head_ID INT
);

CREATE TABLE Instructor (
    Instructor_ID INT PRIMARY KEY,
    Name VARCHAR(150) NOT NULL,
    Email VARCHAR(150),
    Specialization VARCHAR(150),
    Department_ID INT,
```

```
   FOREIGN KEY (Department_ID) REFERENCES
Department(Department_ID)
);

ALTER TABLE Department ADD CONSTRAINT fk_dept_head
FOREIGN KEY (Head_ID) REFERENCES
Instructor(Instructor_ID);

CREATE TABLE Student (
   Student_ID INT PRIMARY KEY,
   Name VARCHAR(150) NOT NULL,
   Email VARCHAR(150),
   Year VARCHAR(10),
   Phone VARCHAR(20),
   Department_ID INT,
   FOREIGN KEY (Department_ID) REFERENCES
Department(Department_ID)
);

CREATE TABLE Classroom (
   Room_ID INT PRIMARY KEY,
   Location VARCHAR(150),
   Capacity INT NOT NULL
);

CREATE TABLE Course (
   Course_ID VARCHAR(20) PRIMARY KEY,
   Course_Name VARCHAR(255) NOT NULL,
   Credits INT NOT NULL,
   Department_ID INT,
   Semester_Offered VARCHAR(50),
   Prerequisite_ID VARCHAR(20) DEFAULT NULL,
   FOREIGN KEY (Department_ID) REFERENCES
Department(Department_ID),
   CONSTRAINT fk_course_prereq FOREIGN KEY
(Prerequisite_ID) REFERENCES Course(Course_ID)
);
```

```
CREATE TABLE Course_Schedule (
   Schedule_ID INT PRIMARY KEY,
   Course_ID VARCHAR(20) NOT NULL,
   Instructor_ID INT NOT NULL,
   Room_ID INT NOT NULL,
   Day VARCHAR(50),
   Time VARCHAR(50),
   FOREIGN KEY (Course_ID) REFERENCES
Course(Course_ID),
   FOREIGN KEY (Instructor_ID) REFERENCES
Instructor(Instructor_ID),
   FOREIGN KEY (Room_ID) REFERENCES
Classroom(Room_ID)
);

CREATE TABLE Registration (
   Reg_ID INT AUTO_INCREMENT PRIMARY KEY,
   Student_ID INT NOT NULL,
   Schedule_ID INT NOT NULL,
   Semester VARCHAR(20) NOT NULL,
   Grade CHAR(2) DEFAULT NULL,
   FOREIGN KEY (Student_ID) REFERENCES
Student(Student_ID),
   FOREIGN KEY (Schedule_ID) REFERENCES
Course_Schedule(Schedule_ID)
);

CREATE TABLE Payment (
   Payment_ID INT PRIMARY KEY,
   Reg_ID INT NOT NULL,
   Amount DECIMAL(10,2),
   Payment_Date DATE,
   Method VARCHAR(50),
   FOREIGN KEY (Reg_ID) REFERENCES Registration(Reg_ID)
);
```

```
CREATE TABLE GradeChangeAudit (
   Log_ID INT AUTO_INCREMENT PRIMARY KEY,
   Reg_ID INT,
   Student_ID INT,
   Old_Grade CHAR(2),
   New_Grade CHAR(2),
   Change_Timestamp TIMESTAMP DEFAULT
CURRENT_TIMESTAMP
);
```

CRUD Operations:

- CREATE:

```
-- Inserting a Department
INSERT INTO Department (Department_ID, Dept_Name)
VALUES (101, 'Computer Science (CSE)');

-- Inserting an Instructor
INSERT INTO Instructor (Instructor_ID, Name, Specialization,
Department_ID) VALUES (201, 'Dr. Priya Sharma', 'Database
Systems', 101);

-- Inserting a Student
INSERT INTO Student (Student_ID, Name, Email, Year,
Department_ID, Phone) VALUES (1001, 'Vikram Singh',
'vicky_singh@uni.edu', 3, 101, '555-0101');

-- Inserting a Registration (handled by the application)
INSERT INTO Registration (Student_ID, Schedule_ID,
Semester) VALUES (1002, 904, 'Semester 6');
```

- READ:

```
-- Read all courses from a specific department
SELECT * FROM Course WHERE Department_ID = 101;

-- Read all registrations for a specific student
SELECT * FROM Registration WHERE Student_ID = 1001;
```

- UPDATE:

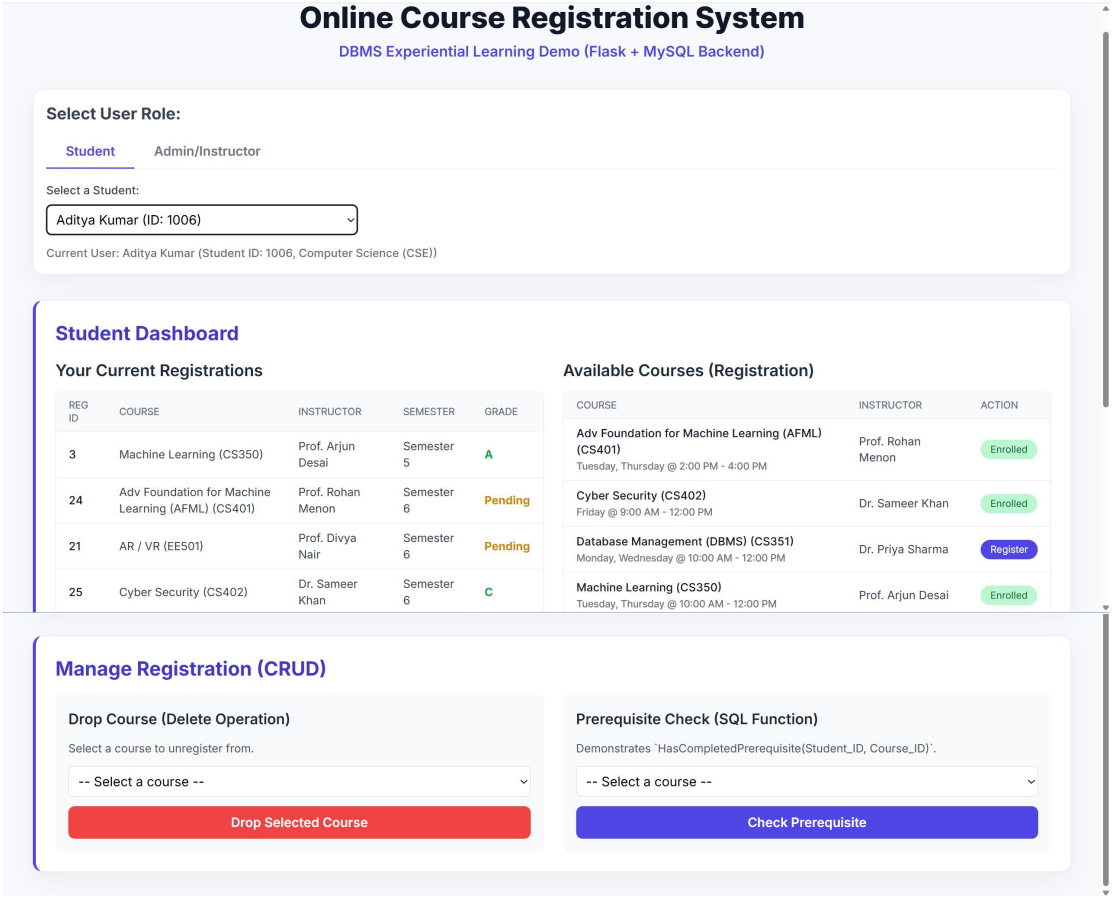
```
-- Update a student's phone number
UPDATE Student SET Phone = '555-1234' WHERE
Student_ID = 1001;

-- Update a student's grade (handled by a stored procedure)
UPDATE Registration SET Grade = 'A' WHERE Reg_ID = 4;
```

- DELETE:

```
-- Delete a registration (dropping a course)
DELETE FROM Registration WHERE Reg_ID = 5 AND
Student_ID = 1001;
```

FRONT END FEATURES:



Online Course Registration System

DBMS Experiential Learning Demo (Flask + MySQL Backend)

Select User Role:

Student | Admin/Instructor

Select an Admin/Instructor:

Dr. Priya Sharma (ID: 201)

Current User: Dr. Priya Sharma (Instructor/Admin ID: 201, Database Systems)

Admin: Grade Management (Update & Procedure)

Demonstrates CRUD Update operation and invokes the `UpdateStudentGrade` Stored Procedure.

Registration ID (Reg_ID): [4] Vikram Singh - Database Management (DB

New Grade: S

Update Grade (Run Procedure)

Admin: Complex Queries & Database Objects

Join Query Demo: Students & Payments

Find students who have completed payment for a registration.

Run Join Query

Nested Query Demo: High Achievers

Find students enrolled in courses taught by Dr. Priya Sharma.

Run Nested Query

Aggregate Query Demo: Departmental Load

Count the total credits offered per Department.

Run Aggregate Query

Query/Log Output

```
Query results will appear here...
```

Trigger Log (`GradeChangeAudit`)

LOG ID	REG ID	STUDENT ID	OLD GRADE	NEW GRADE	TIMESTAMP
14	31	1003	NULL	S	2025-11-07 10:39:15
13	5	1001	B	S	2025-11-07 10:34:54
12	29	1001	NULL	B	2025-11-07 10:33:56
11	4	1001	B	C	2025-11-07 10:33:32
10	4	1001	S	B	2025-11-07 08:19:44

TRIGGERS:

```
DELIMITER $$
CREATE TRIGGER trg_AfterGradeUpdate
AFTER UPDATE ON Registration
FOR EACH ROW
BEGIN
   IF (OLD.Grade IS NULL AND NEW.Grade IS NOT NULL)
      OR (OLD.Grade IS NOT NULL AND NEW.Grade IS NULL)
      OR (OLD.Grade <> NEW.Grade) THEN
       INSERT INTO GradeChangeAudit (Reg_ID, Student_ID,
Old_Grade, New_Grade)
       VALUES (OLD.Reg_ID, OLD.Student_ID, OLD.Grade,
NEW.Grade);
   END IF;
END$$
DELIMITER ;
```

PROCEDURES:

```
DELIMITER $$
CREATE PROCEDURE UpdateStudentGrade(IN regId INT, IN
newGrade CHAR(2))
BEGIN
   UPDATE Registration
   SET Grade = newGrade
   WHERE Reg_ID = regId;
END$$
DELIMITER ;
```

FUNCTIONS:

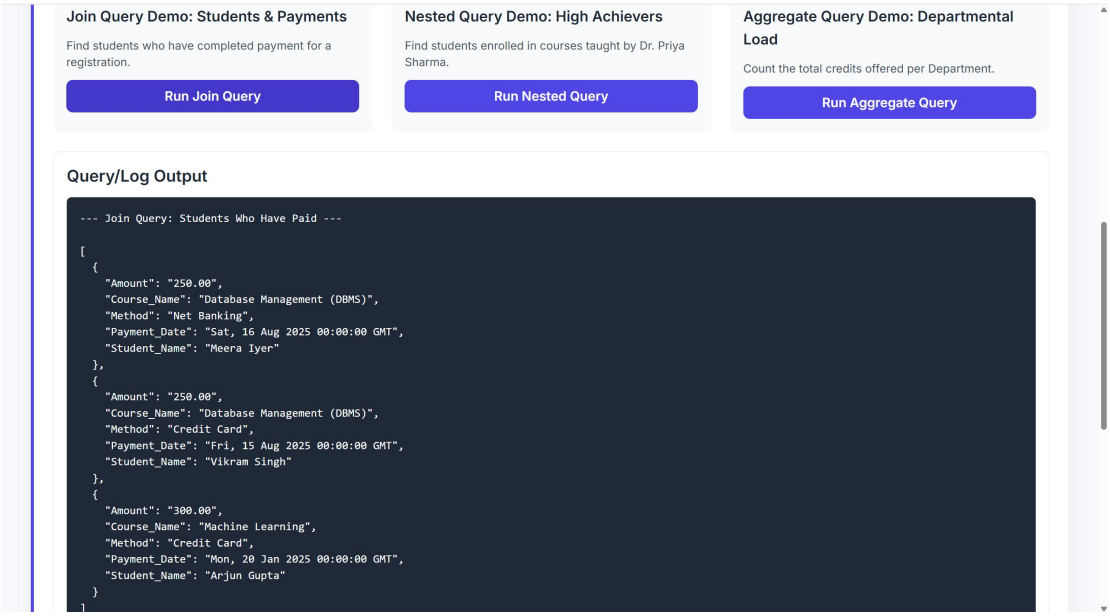
GetAvailableSeats(schedule_id): Returns the number of remaining seats for a given course schedule by subtracting the count of enrolled students from the classroom's capacity.

HasCompletedPrerequisite(student_id, course_id): Returns 1 (true) if the student has a passing grade for the specified course's prerequisite, and 0 (false) otherwise.

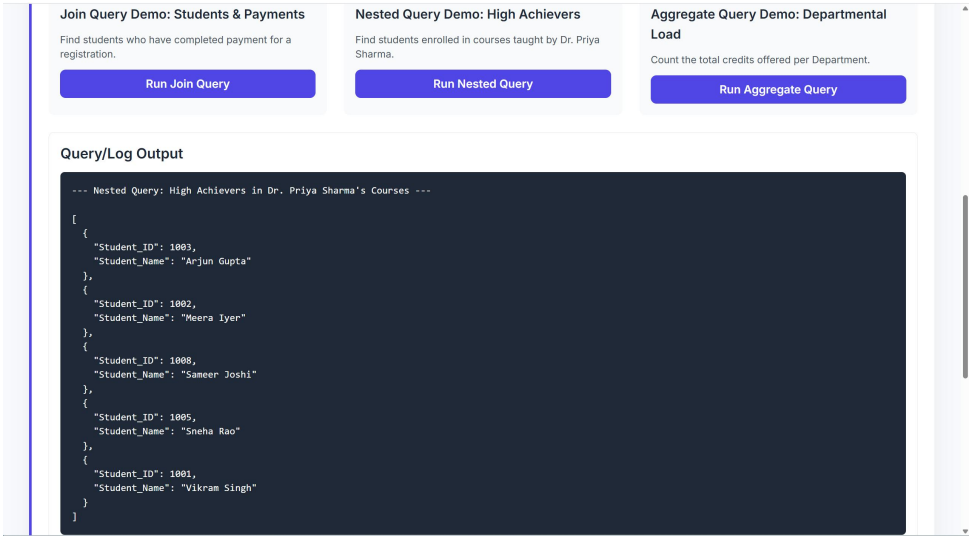
NESTED, JOINS AND AGGREGATE QUERY:

The application demonstrates several complex queries, including:

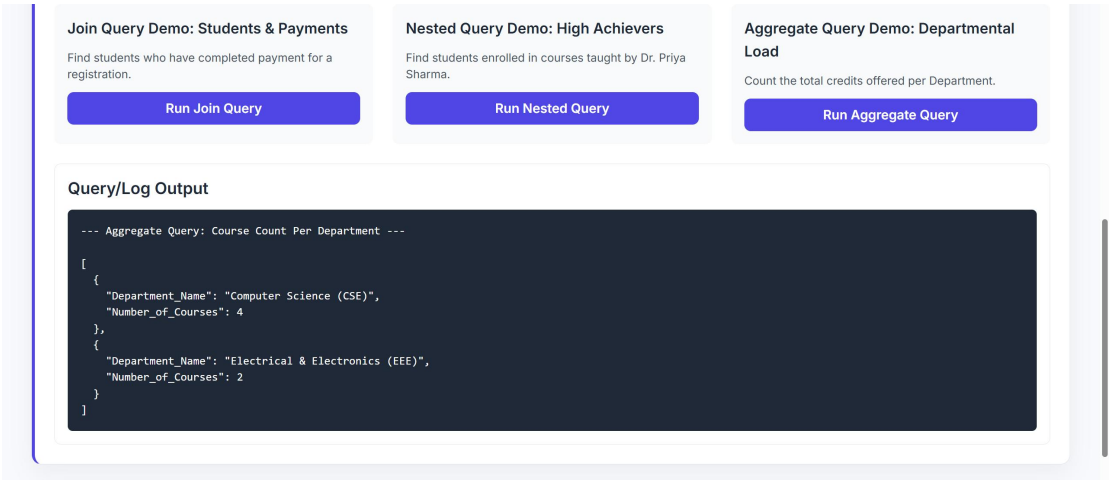
1. Join Query: Retrieves student names, course names, and payment details by
   joining Student, Registration, Payment, Course_Schedule, and Course tables.



2. Nested Query: Finds all students who have achieved high grades in courses taught by a specific instructor, using a WHERE IN (...) subquery.



3. Aggregate Query: Calculates the total number of courses offered per department using COUNT and GROUP BY.



INVOKE TRIGGERS, FUNCTIONS AND PROCEDURES

-- Invoke a procedure to update a grade
-- This action will automatically fire the trg_AfterGradeUpdate trigger.

```
CALL UpdateStudentGrade(5, 'B');

-- Invoke a function to check available seats before
registration
SELECT GetAvailableSeats(901) AS 'Seats Available';

-- Invoke a function to check for prerequisites
SELECT HasCompletedPrerequisite(1001, 'CS401') AS
'Prerequisite Met?';

-- A direct UPDATE will also fire the trigger
UPDATE Registration SET Grade = 'C' WHERE Reg_ID = 5;
```

Trigger Log (`GradeChangeAudit`)

LOG ID	REG ID	STUDENT ID	OLD GRADE	NEW GRADE	TIMESTAMP
14	31	1003	NULL	S	2025-11-07 10:39:15
13	5	1001	B	S	2025-11-07 10:34:54
12	29	1001	NULL	B	2025-11-07 10:33:56
11	4	1001	B	C	2025-11-07 10:33:32
10	4	1001	S	B	2025-11-07 08:19:44
9	4	1001	B	S	2025-11-07 08:18:11
8	4	1001	D	B	2025-11-07 08:15:53
7	4	1001	A	D	2025-11-07 02:33:56
6	26	1001	D	A	2025-11-07 02:33:12
5	26	1001	NULL	D	2025-11-07 02:33:02