PROJECT ON

"STUDENT MANAGEMENT SYSTEM"

SUBJECT : DATABASE MANAGEMENT SYSTEMS

By :

Abel Biju

CSE A

00417702723

Submitted to: Prof. Vidushi

Introduction

The Student Management System represents a comprehensive database solution designed to address the administrative challenges faced by educational institutions. This project implements a relational database using MySQL to efficiently organize and manage core academic operations including student registration, course enrolment, attendance tracking, and grade management.

At its foundation, the system employs robust database principles including entity-relationship modelling and normalization to Boyce-Codd Normal Form (BCNF), ensuring optimal data integrity and elimination of redundancies. The implementation features essential database components such as stored procedures for enrolment processing, triggers for data validation, and views for simplified reporting.

This solution demonstrates how relational database technology can transform traditional paper-based academic administration into an efficient, automated process. By centralizing student information and academic records, the system provides institutions with accurate, real-time data access while maintaining strict data consistency through carefully designed constraints and relationships.

The following sections detail the system's architecture, from its normalized table structure to the practical implementation of database objects that collectively deliver a functional management platform suitable for educational environments of varying scales. The project not only serves as an academic exercise in database design but also presents a viable framework for actual institutional deployment.

Objectives:

1. Design a **normalized database (BCNF)** to store student, course, and faculty data without redundancy.
2. Automate key processes like **enrollment** and **grading** using stored procedures.
3. Ensure **data integrity** with constraints and triggers (e.g., email validation).
4. Generate **real-time reports** (attendance, grades) via SQL views.
5. Provide a **scalable foundation** for academic institutions to manage records efficiently.
6. Demonstrate **real-world DBMS applications** through queries and transactions.

System Architecture

The system is designed with a modular three-tier architecture comprising the following layers:

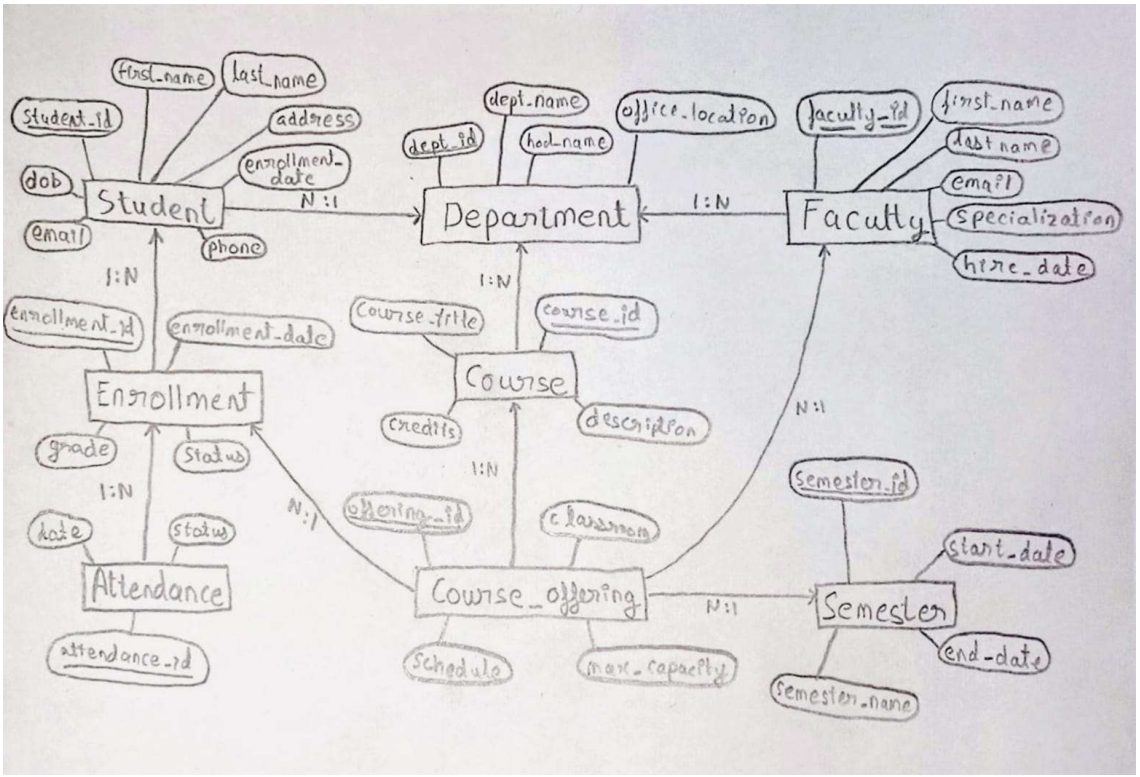
- Presentation Layer (SQL client interface like MySQL Workbench)
- Logic Layer (Stored procedures, triggers, SQL scripts)
- Data Layer (MySQL database with relational tables)

SQL Features Used

SQL was used to create and manipulate the database with the following features:

- DDL: CREATE TABLE with constraints (PRIMARY KEY, FOREIGN KEY, NOT NULL, CHECK).
- DML: INSERT, UPDATE, DELETE for data operations.
- Normalization: All tables have been normalized to BCNF(Boyce-Codd Normal Form)
- Joins: INNER, LEFT, RIGHT JOINs for fetching combined data.
- Subqueries: Used in views and stored procedures.
- Aggregate Functions: SUM, AVG, COUNT used in reports.
- Views: For admin reporting on customer orders.
- Triggers: Automatically update stock when an order is placed.
- Stored Procedures: Handle order placement, price calculations, and inventory updates.

ER-DIAGRAM


Student
Department
Faculty
Enrollment
Course
Course_offering
Attendance
Semester
student_id
first_name
last_name
address
dob
email
phone
enrollment_date
N:1
dept_id
dept_name
hod_name
office_location
faculty_id
first_name
last_name
email
specialization
hire_date
1:N
enrollment_id
enrollment_date
grade
status
course_title
course_id
credits
description
offering_id
classroom
schedule
max_capacity
semester_id
start_date
end_date
semester_name
date
status
attendance_id

Tables :

1. Department Table

Datatype	Attribute Name	Constraints
VARCHAR(5)	dept_id	PRIMARY KEY
VARCHAR(50)	dept_name	NOT NULL, UNIQUE
VARCHAR(100)	hod_name	
VARCHAR(50)	office_location	
VARCHAR(15)	contact_phone	
DATE	established_date	

2. Faculty Table

Datatype	Attribute Name	Constraints
VARCHAR(10)	faculty_id	PRIMARY KEY
VARCHAR(50)	first_name	NOT NULL
VARCHAR(50)	last_name	NOT NULL
VARCHAR(5)	dept_id	NOT NULL, FK → Department(dept_id)
VARCHAR(100)	email	NOT NULL, UNIQUE
VARCHAR(15)	phone	
DATE	hire_date	
VARCHAR(100)	specialization	

3. Student Table

Datatype	Attribute Name	Constraints
VARCHAR(10)	student_id	PRIMARY KEY
VARCHAR(50)	first_name	NOT NULL
VARCHAR(50)	last_name	NOT NULL
VARCHAR(5)	dept_id	NOT NULL, FK → Department(dept_id)
VARCHAR(100)	email	NOT NULL, UNIQUE
VARCHAR(15)	phone	
DATE	dob	
TEXT	address	
DATE	enrollment_date	

4. Course Table

Datatype	Attribute Name	Constraints
VARCHAR(10)	course_id	PRIMARY KEY
VARCHAR(100)	course_title	NOT NULL
VARCHAR(5)	dept_id	NOT NULL, FK → Department(dept_id)
INT	credits	NOT NULL, CHECK (credits > 0)
TEXT	description	

5. Semester Table

Datatype	Attribute Name	Constraints
VARCHAR(10)	semester_id	PRIMARY KEY
VARCHAR(20)	semester_name	NOT NULL
DATE	start_date	NOT NULL
DATE	end_date	NOT NULL, CHECK (end_date > start_date)

6. CourseOffering Table

Datatype	Attribute Name	Constraints
VARCHAR(15)	offering_id	PRIMARY KEY
VARCHAR(10)	course_id	NOT NULL, FK → Course(course_id)
VARCHAR(10)	semester_id	NOT NULL, FK → Semester(semester_id)
VARCHAR(10)	faculty_id	NOT NULL, FK → Faculty(faculty_id)
VARCHAR(20)	classroom	
VARCHAR(50)	schedule	
INT	max_capacity	
		UNIQUE (course_id, semester_id, faculty_id)

7. Enrollment Table

Datatype	Attribute Name	Constraints
VARCHAR(15)	enrollment_id	PRIMARY KEY
VARCHAR(10)	student_id	NOT NULL, FK → Student(student_id)
VARCHAR(15)	offering_id	NOT NULL, FK → CourseOffering(offering_id)
DATE	enrollment_date	NOT NULL
VARCHAR(2)	grade	CHECK (grade IN ('A','A-','B+',...,'F','I','W'))
VARCHAR(20)	status	DEFAULT 'Active'
		UNIQUE (student_id, offering_id)

8. Attendance Table

Datatype	Attribute Name	Constraints
INT	attendance_id	PRIMARY KEY, AUTO_INCREMENT
VARCHAR(15)	enrollment_id	NOT NULL, FK → Enrollment(enrollment_id)
DATE	date	NOT NULL
ENUM	status	NOT NULL ('Present','Absent','Late')
		UNIQUE (enrollment_id, date)

9. EnrollmentAudit Table

Datatype	Attribute Name	Constraints
INT	audit_id	PRIMARY KEY, AUTO_INCREMENT
VARCHAR(15)	enrollment_id	
VARCHAR(10)	student_id	
VARCHAR(15)	offering_id	
VARCHAR(10)	action_type	
DATETIME	action_date	
VARCHAR(50)	changed_by	

Implementation

```
-- Database creation
DROP DATABASE IF EXISTS StudentManagementSystem;
CREATE DATABASE StudentManagementSystem;
USE StudentManagementSystem;
-- 1. Department Table
CREATE TABLE Department (
    dept_id VARCHAR(5) PRIMARY KEY,
    dept_name VARCHAR(50) NOT NULL UNIQUE,
    hod_name VARCHAR(100),
    office_location VARCHAR(50),
    contact_phone VARCHAR(15),
    established_date DATE
);
-- 2. Faculty Table
CREATE TABLE Faculty (
    faculty_id VARCHAR(10) PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    dept_id VARCHAR(5) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    phone VARCHAR(15),
    hire_date DATE,
    specialization VARCHAR(100),
    FOREIGN KEY (dept_id) REFERENCES Department(dept_id)
);
-- 3. Student Table
CREATE TABLE Student (
    student_id VARCHAR(10) PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    dept_id VARCHAR(5) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    phone VARCHAR(15),
    dob DATE,
    address TEXT,
    enrollment_date DATE,
    FOREIGN KEY (dept_id) REFERENCES Department(dept_id)
);
-- 4. Course Table
CREATE TABLE Course (
    course_id VARCHAR(10) PRIMARY KEY,
    course_title VARCHAR(100) NOT NULL,
    dept_id VARCHAR(5) NOT NULL,
    credits INT NOT NULL CHECK (credits > 0),
    description TEXT,
    FOREIGN KEY (dept_id) REFERENCES Department(dept_id)
);
-- 5. Semester Table
CREATE TABLE Semester (
    semester_id VARCHAR(10) PRIMARY KEY,
    semester_name VARCHAR(20) NOT NULL,
    start_date DATE NOT NULL,
    end_date DATE NOT NULL,
    CONSTRAINT date_check CHECK (end_date > start_date)
);
-- 6. CourseOffering Table
CREATE TABLE CourseOffering (
    offering_id VARCHAR(15) PRIMARY KEY,
```

```
    course_id VARCHAR(10) NOT NULL,
    semester_id VARCHAR(10) NOT NULL,
    faculty_id VARCHAR(10) NOT NULL,
    classroom VARCHAR(20),
    schedule VARCHAR(50),
    max_capacity INT,
    FOREIGN KEY (course_id) REFERENCES Course(course_id),
    FOREIGN KEY (semester_id) REFERENCES Semester(semester_id),
    FOREIGN KEY (faculty_id) REFERENCES Faculty(faculty_id),
    CONSTRAINT unique_offering UNIQUE (course_id, semester_id, faculty_id)
);
-- 7. Enrollment Table
CREATE TABLE Enrollment (
    enrollment_id VARCHAR(15) PRIMARY KEY,
    student_id VARCHAR(10) NOT NULL,
    offering_id VARCHAR(15) NOT NULL,
    enrollment_date DATE NOT NULL,
    grade VARCHAR(2),
    status VARCHAR(20) DEFAULT 'Active',
    FOREIGN KEY (student_id) REFERENCES Student(student_id),
    FOREIGN KEY (offering_id) REFERENCES CourseOffering(offering_id),
    CONSTRAINT unique_enrollment UNIQUE (student_id, offering_id),
    CONSTRAINT valid_grade CHECK (grade IN ('A', 'A-', 'B+', 'B', 'B-', 'C+', 'C', 'C-', 'D+', 'D', 'F', 'I', 'W',
NULL))
);
-- 8. Attendance Table
CREATE TABLE Attendance (
    attendance_id INT AUTO_INCREMENT PRIMARY KEY,
    enrollment_id VARCHAR(15) NOT NULL,
    date DATE NOT NULL,
    status ENUM('Present', 'Absent', 'Late') NOT NULL,
    FOREIGN KEY (enrollment_id) REFERENCES Enrollment(enrollment_id),
    CONSTRAINT unique_attendance UNIQUE (enrollment_id, date)
);
-- 9. Enrollment Audit Table (for trigger)
CREATE TABLE EnrollmentAudit (
    audit_id INT AUTO_INCREMENT PRIMARY KEY,
    enrollment_id VARCHAR(15),
    student_id VARCHAR(10),
    offering_id VARCHAR(15),
    action_type VARCHAR(10),
    action_date DATETIME,
    changed_by VARCHAR(50)
);
-- Views
-- 1. Student Course View
CREATE VIEW StudentCourseView AS
SELECT
    s.student_id,
    CONCAT(s.first_name, ' ', s.last_name) AS student_name,
    c.course_id,
    c.course_title,
    co.schedule,
    f.faculty_id,
    CONCAT(f.first_name, ' ', f.last_name) AS faculty_name,
    e.grade
FROM
    Student s
JOIN
    Enrollment e ON s.student_id = e.student_id
JOIN
```

```
    CourseOffering co ON e.offering_id = co.offering_id
JOIN
    Course c ON co.course_id = c.course_id
JOIN
    Faculty f ON co.faculty_id = f.faculty_id;
-- 2. Department Summary View
CREATE VIEW DepartmentSummary AS
SELECT
    d.dept_id,
    d.dept_name,
    COUNT(DISTINCT s.student_id) AS student_count,
    COUNT(DISTINCT f.faculty_id) AS faculty_count,
    COUNT(DISTINCT c.course_id) AS course_count
FROM
    Department d
LEFT JOIN
    Student s ON d.dept_id = s.dept_id
LEFT JOIN
    Faculty f ON d.dept_id = f.dept_id
LEFT JOIN
    Course c ON d.dept_id = c.dept_id
GROUP BY
    d.dept_id, d.dept_name;
-- Stored Procedures
DELIMITER //
-- 1. Enroll Student Procedure
CREATE PROCEDURE EnrollStudent(
    IN p_student_id VARCHAR(10),
    IN p_offering_id VARCHAR(15),
    OUT p_result VARCHAR(100))
BEGIN
    DECLARE student_exists INT;
    DECLARE offering_exists INT;
    DECLARE already_enrolled INT;
    DECLARE capacity INT;
    DECLARE current_enrollment INT;
    SELECT COUNT(*) INTO student_exists FROM Student WHERE student_id = p_student_id;
    SELECT COUNT(*) INTO offering_exists FROM CourseOffering WHERE offering_id = p_offering_id;
    SELECT COUNT(*) INTO already_enrolled FROM Enrollment
    WHERE student_id = p_student_id AND offering_id = p_offering_id;
    SELECT max_capacity INTO capacity FROM CourseOffering WHERE offering_id = p_offering_id;
    SELECT COUNT(*) INTO current_enrollment FROM Enrollment WHERE offering_id = p_offering_id;
    IF student_exists = 0 THEN
        SET p_result = 'Error: Student does not exist';
    ELSEIF offering_exists = 0 THEN
        SET p_result = 'Error: Course offering does not exist';
    ELSEIF already_enrolled > 0 THEN
        SET p_result = 'Error: Student already enrolled in this course';
    ELSEIF capacity IS NOT NULL AND current_enrollment >= capacity THEN
        SET p_result = 'Error: Course has reached maximum capacity';
    ELSE
        INSERT INTO Enrollment (enrollment_id, student_id, offering_id, enrollment_date)
        VALUES (CONCAT('ENR', UUID_SHORT()), p_student_id, p_offering_id, CURDATE());
        SET p_result = 'Success: Student enrolled successfully';
    END IF;
END //
-- 2. Update Grade Procedure
CREATE PROCEDURE UpdateGrade(
    IN p_enrollment_id VARCHAR(15),
    IN p_grade VARCHAR(2),
    OUT p_result VARCHAR(100))
```

```
BEGIN
    DECLARE valid_grade INT DEFAULT 0;
    IF p_grade IN ('A', 'A-', 'B+', 'B', 'B-', 'C+', 'C', 'C-', 'D+', 'D', 'F', 'I', 'W') THEN
        SET valid_grade = 1;
    END IF;
    IF NOT EXISTS (SELECT 1 FROM Enrollment WHERE enrollment_id = p_enrollment_id) THEN
        SET p_result = 'Error: Enrollment record not found';
    ELSEIF valid_grade = 0 THEN
        SET p_result = 'Error: Invalid grade provided';
    ELSE
        UPDATE Enrollment SET grade = p_grade WHERE enrollment_id = p_enrollment_id;
        SET p_result = CONCAT('Success: Grade updated to ', p_grade);
    END IF;
END //
-- Triggers
-- 1. Enrollment Audit Triggers
CREATE TRIGGER after_enrollment_insert
AFTER INSERT ON Enrollment
FOR EACH ROW
BEGIN
    INSERT INTO EnrollmentAudit (enrollment_id, student_id, offering_id, action_type, action_date,
changed_by)
    VALUES (NEW.enrollment_id, NEW.student_id, NEW.offering_id, 'INSERT', NOW(), CURRENT_USER());
END //
CREATE TRIGGER after_enrollment_update
AFTER UPDATE ON Enrollment
FOR EACH ROW
BEGIN
    INSERT INTO EnrollmentAudit (enrollment_id, student_id, offering_id, action_type, action_date,
changed_by)
    VALUES (NEW.enrollment_id, NEW.student_id, NEW.offering_id, 'UPDATE', NOW(), CURRENT_USER());
END //
CREATE TRIGGER after_enrollment_delete
AFTER DELETE ON Enrollment
FOR EACH ROW
BEGIN
    INSERT INTO EnrollmentAudit (enrollment_id, student_id, offering_id, action_type, action_date,
changed_by)
    VALUES (OLD.enrollment_id, OLD.student_id, OLD.offering_id, 'DELETE', NOW(), CURRENT_USER());
END //
-- 2. Student Email Validation Triggers
CREATE TRIGGER before_student_insert
BEFORE INSERT ON Student
FOR EACH ROW
BEGIN
    IF NEW.email NOT LIKE '%@%.%' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Invalid email format';
    END IF;
END //
CREATE TRIGGER before_student_update
BEFORE UPDATE ON Student
FOR EACH ROW
BEGIN
    IF NEW.email NOT LIKE '%@%.%' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Invalid email format';
    END IF;
END //
DELIMITER ;
```

```
-- 1. Department Table
INSERT INTO Department (dept_id, dept_name, hod_name, office_location, contact_phone,
established_date) VALUES
('CS', 'Computer Science', 'Dr. Rajesh Kumar', 'Block A, Room 101', '011-27894561', '2005-07-15'),
('MATH', 'Mathematics', 'Dr. Priya Sharma', 'Block B, Room 205', '011-27894562', '1998-06-20'),
('PHY', 'Physics', 'Dr. Amit Patel', 'Block C, Room 310', '011-27894563', '2001-03-10'),
('CHEM', 'Chemistry', 'Dr. Neha Gupta', 'Block D, Room 415', '011-27894564', '2003-09-05'),
('ENG', 'English', 'Dr. Sanjay Verma', 'Block E, Room 520', '011-27894565', '1995-11-12'),
('ECE', 'Electronics', 'Dr. Anil Kapoor', 'Block F, Room 615', '011-27894566', '2007-08-25'),
('BIO', 'Biology', 'Dr. Meera Singh', 'Block G, Room 720', '011-27894567', '2002-04-18'),
('ECO', 'Economics', 'Dr. Vivek Malhotra', 'Block H, Room 825', '011-27894568', '1999-10-30');

-- 2. Faculty Table
INSERT INTO Faculty (faculty_id, first_name, last_name, dept_id, email, phone, hire_date, specialization)
VALUES
('CS101', 'Anjali', 'Singh', 'CS', 'asingh@univ.edu', '9876543210', '2012-08-15', 'Artificial Intelligence'),
('CS102', 'Vikram', 'Joshi', 'CS', 'vjoshi@univ.edu', '9876543211', '2015-06-20', 'Database Systems'),
('MATH201', 'Rahul', 'Mehta', 'MATH', 'rmehta@univ.edu', '9876543212', '2010-07-10', 'Applied
Mathematics'),
('PHY301', 'Deepika', 'Reddy', 'PHY', 'dreddy@univ.edu', '9876543213', '2018-01-05', 'Quantum Physics'),
('CHEM401', 'Arun', 'Malhotra', 'CHEM', 'amalhotra@univ.edu', '9876543214', '2016-03-15', 'Organic
Chemistry'),
('ENG501', 'Priyanka', 'Chopra', 'ENG', 'pchopra@univ.edu', '9876543215', '2014-09-22', 'Modern
Literature'),
('ECE601', 'Ravi', 'Shastri', 'ECE', 'rshastri@univ.edu', '9876543216', '2017-11-08', 'Embedded Systems'),
('BIO701', 'Sunita', 'Rao', 'BIO', 'srao@univ.edu', '9876543217', '2019-05-14', 'Genetics');

-- 3. Student Table
INSERT INTO Student (student_id, first_name, last_name, dept_id, email, phone, dob, address,
enrollment_date) VALUES
('S1001', 'Rohan', 'Sharma', 'CS', 'rohan.sharma@univ.edu', '9876543301', '2000-05-15', '12, Green Park,
New Delhi', '2021-07-01'),
('S1002', 'Priya', 'Verma', 'MATH', 'priya.verma@univ.edu', '9876543302', '2001-02-20', '34, Saket, New
Delhi', '2021-07-01'),
('S1003', 'Amit', 'Kumar', 'PHY', 'amit.kumar@univ.edu', '9876543303', '2000-11-10', '56, Rohini, New
Delhi', '2021-07-01'),
('S1004', 'Neha', 'Gupta', 'CHEM', 'neha.gupta@univ.edu', '9876543304', '2001-03-25', '78, Dwarka, New
Delhi', '2021-07-01'),
('S1005', 'Sanjay', 'Patel', 'ENG', 'sanjay.patel@univ.edu', '9876543305', '2000-09-12', '90, Vasant Kunj,
New Delhi', '2021-07-01'),
('S1006', 'Ananya', 'Reddy', 'ECE', 'ananya.reddy@univ.edu', '9876543306', '2001-07-18', '112, Lajpat
Nagar, New Delhi', '2021-07-01'),
('S1007', 'Vikrant', 'Singh', 'BIO', 'vikrant.singh@univ.edu', '9876543307', '2000-12-05', '134, Hauz Khas,
New Delhi', '2021-07-01'),
('S1008', 'Mehak', 'Agarwal', 'ECO', 'mehak.agarwal@univ.edu', '9876543308', '2001-04-30', '156,
Mayur Vihar, New Delhi', '2021-07-01');

-- 4. Course Table
INSERT INTO Course (course_id, course_title, dept_id, credits, description) VALUES
('CS101', 'Introduction to Programming', 'CS', 4, 'Fundamentals of programming using Python'),
('CS201', 'Database Systems', 'CS', 4, 'Relational database design and implementation'),
('MATH101', 'Calculus I', 'MATH', 3, 'Differential and integral calculus'),
('PHY101', 'Classical Mechanics', 'PHY', 3, 'Newtonian mechanics and its applications'),
('CHEM101', 'Organic Chemistry', 'CHEM', 3, 'Structure and reactions of organic compounds'),
('ENG101', 'English Literature', 'ENG', 2, 'Survey of English literature from Chaucer to present'),
('ECE101', 'Circuit Theory', 'ECE', 4, 'Basic electrical circuits analysis'),
('BIO101', 'Cell Biology', 'BIO', 3, 'Structure and function of cells');

-- 5. Semester Table
INSERT INTO Semester (semester_id, semester_name, start_date, end_date) VALUES
('FALL2022', 'Fall 2022', '2022-08-01', '2022-12-15'),
```

```
('SPRING2023', 'Spring 2023', '2023-01-10', '2023-05-20'),
('FALL2023', 'Fall 2023', '2023-08-01', '2023-12-15'),
('SPRING2024', 'Spring 2024', '2024-01-10', '2024-05-20'),
('SUMMER2024', 'Summer 2024', '2024-06-01', '2024-07-31'),
('FALL2024', 'Fall 2024', '2024-08-01', '2024-12-15'),
('SPRING2025', 'Spring 2025', '2025-01-10', '2025-05-20'),
('SUMMER2025', 'Summer 2025', '2025-06-01', '2025-07-31');

-- 6. CourseOffering Table
INSERT INTO CourseOffering (offering_id, course_id, semester_id, faculty_id, classroom, schedule,
max_capacity) VALUES
('CS101-F22', 'CS101', 'FALL2022', 'CS101', 'A101', 'Mon/Wed 10:00-11:30', 30),
('CS201-S23', 'CS201', 'SPRING2023', 'CS102', 'A102', 'Tue/Thu 14:00-15:30', 25),
('MATH101-F22', 'MATH101', 'FALL2022', 'MATH201', 'B201', 'Mon/Wed/Fri 09:00-10:00', 40),
('PHY101-S23', 'PHY101', 'SPRING2023', 'PHY301', 'C301', 'Tue/Thu 11:00-12:30', 35),
('CHEM101-F23', 'CHEM101', 'FALL2023', 'CHEM401', 'D401', 'Mon/Wed 13:00-14:30', 30),
('ENG101-S24', 'ENG101', 'SPRING2024', 'ENG501', 'E501', 'Fri 10:00-13:00', 20),
('ECE101-F24', 'ECE101', 'FALL2024', 'ECE601', 'F601', 'Tue/Thu 09:00-10:30', 25),
('BIO101-S25', 'BIO101', 'SPRING2025', 'BIO701', 'G701', 'Mon/Wed 15:00-16:30', 30);

-- 7. Enrollment Table
INSERT INTO Enrollment (enrollment_id, student_id, offering_id, enrollment_date, grade, status) VALUES
('ENR1001', 'S1001', 'CS101-F22', '2022-08-05', 'A', 'Completed'),
('ENR1002', 'S1001', 'CS201-S23', '2023-01-12', 'B+', 'Completed'),
('ENR1003', 'S1002', 'MATH101-F22', '2022-08-05', 'A-', 'Completed'),
('ENR1004', 'S1003', 'PHY101-S23', '2023-01-12', 'B', 'Completed'),
('ENR1005', 'S1004', 'CHEM101-F23', '2023-08-03', NULL, 'Active'),
('ENR1006', 'S1005', 'ENG101-S24', '2024-01-15', NULL, 'Active'),
('ENR1007', 'S1006', 'ECE101-F24', '2024-08-05', NULL, 'Active'),
('ENR1008', 'S1007', 'BIO101-S25', '2025-01-10', NULL, 'Active');

-- 8. Attendance Table
INSERT INTO Attendance (enrollment_id, date, status) VALUES
('ENR1001', '2022-08-08', 'Present'),
('ENR1001', '2022-08-10', 'Present'),
('ENR1001', '2022-08-15', 'Late'),
('ENR1001', '2022-08-17', 'Present'),
('ENR1003', '2022-08-08', 'Present'),
('ENR1003', '2022-08-10', 'Absent'),
('ENR1003', '2022-08-15', 'Present'),
('ENR1005', '2023-08-07', 'Present');

-- 9. EnrollmentAudit Table (will be populated automatically by triggers)
```

REAL LIFE QUERY IMPLEMENTATION

1. **Active Course Enrollment List**

```
SELECT s.student_id,
    CONCAT(s.first_name, ' ', s.last_name) AS student_name,
    c.course_title,
    co.schedule,
    f.first_name AS faculty_first_name
FROM Enrollment e
JOIN Student s ON e.student_id = s.student_id
JOIN CourseOffering co ON e.offering_id = co.offering_id
JOIN Course c ON co.course_id = c.course_id
JOIN Faculty f ON co.faculty_id = f.faculty_id
WHERE e.status = 'Active'
LIMIT 10;
```

student_id	student_name	course_title	schedule	faculty_first_name
S1004	Neha Gupta	Organic Chemistry	Mon/Wed 13:00-14:30	Arun
S1005	Sanjay Patel	English Literature	Fri 10:00-13:00	Priyanka
S1006	Ananya Reddy	Circuit Theory	Tue/Thu 09:00-10:30	Ravi
S1007	Vikrant Singh	Cell Biology	Mon/Wed 15:00-16:30	Sunita

2. **Department-wise Faculty Count (Shows all departments)**

```
SELECT d.dept_name,
    COUNT(f.faculty_id) AS faculty_count
FROM Department d
LEFT JOIN Faculty f ON d.dept_id = f.dept_id
GROUP BY d.dept_id
ORDER BY faculty_count DESC;
```

dept_name	faculty_count
Computer Science	2
Chemistry	1
Biology	1
Physics	1
Mathematics	1
English	1
Electronics	1
Economics	0

3. **Course Popularity Ranking (Uses existing offerings)**

```
SELECT c.course_id, c.course_title,
    COUNT(e.enrollment_id) AS enrollment_count
FROM CourseOffering co
JOIN Course c ON co.course_id = c.course_id
LEFT JOIN Enrollment e ON co.offering_id = e.offering_id
GROUP BY co.course_id
ORDER BY enrollment_count DESC
LIMIT 5;
```

course_id	course_title	enrollment_count
CS101	Introduction to Programming	1
CHEM101	Organic Chemistry	1
BIO101	Cell Biology	1
PHY101	Classical Mechanics	1
MATH101	Calculus I	1

4. **Count courses taught by each faculty per semester**

```
SELECT f.faculty_id, CONCAT(f.first_name, ' ', f.last_name) AS
faculty_name,
    co.semester_id,
    COUNT(co.offering_id) AS courses_taught,
    SUM(c.credits) AS total_credits
FROM Faculty f
JOIN CourseOffering co ON f.faculty_id = co.faculty_id
JOIN Course c ON co.course_id = c.course_id
GROUP BY f.faculty_id, co.semester_id
ORDER BY co.semester_id, total_credits DESC;
```

faculty_id	faculty_name	semester_id	courses_taught	total_credits
CS101	Anjali Singh	FALL2022	1	4
MATH201	Rahul Mehta	FALL2022	1	3
CHEM401	Arun Malhotra	FALL2023	1	3
ECE601	Ravi Shastri	FALL2024	1	4
CS102	Vikram Joshi	SPRING2023	1	4
PHY301	Deepika Reddy	SPRING2023	1	3
ENG501	Priyanka Chopra	SPRING2024	1	2
BIO701	Sunita Rao	SPRING2025	1	3

5. List of all students.

```
SELECT S.student_id, S.first_name, S.last_name
  from Student as S;
```

student_id	first_name	last_name
S1001	Rohan	Sharma
S1002	Priya	Verma
S1003	Amit	Kumar
S1004	Neha	Gupta
S1005	Sanjay	Patel
S1006	Ananya	Reddy
S1007	Vikrant	Singh
S1008	Mehak	Agarwal

6. List of all faculty.

```
SELECT F.faculty_id, F.first_name, F.last_name
  from Faculty as F;
```

faculty_id	first_name	last_name
BIO701	Sunita	Rao
CHEM401	Arun	Malhotra
CS101	Anjali	Singh
CS102	Vikram	Joshi
ECE601	Ravi	Shastri
ENG501	Priyanka	Chopra
MATH201	Rahul	Mehta
PHY301	Deepika	Reddy

Conclusion

The Student Management System demonstrates how a well-designed database can transform academic administration by automating core processes like enrollment, attendance tracking, and grading. Through normalized tables, stored procedures, and triggers, this project ensures data accuracy while eliminating redundancy—proving that robust database architecture directly translates to operational efficiency. The implementation of real-world queries for attendance marking, transcript generation, and enrollment analytics showcases the system's practical value for educators, administrators, and students alike. As educational institutions increasingly adopt digital solutions, this project serves as both a functional prototype and a learning model for implementing scalable, maintainable database systems in academic environments. Future enhancements could integrate user authentication or mobile access, but the current foundation already delivers a significant upgrade over manual record-keeping. By bridging database theory with institutional needs, this system highlights the critical role of DBMS in modern education management.