THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY

(Deemed to be University)
Patiala, Punjab

A Mini Project On

"University Management System"

For the partial complement of Database Management System 2025
Under the supervision of
"Dr. Shashank Singh"
Department of Computer Science and Engineering

Submitted By:

"Sajid Miya" **102367013**

Submitted To:

THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY
Department of Computer Science and Engineering
Patiala, Punjab, India
"May 2025"

Table of Contents

1. INTRODUCTION 1
1.1 Background 1
1.2 Problem Statement 1
1.3 Objectives 1
1.4 Scope of the Project 2
1.4.1 Functional Scope 2
1.4.2. Technical Scope 3
1.4.3. Limitations (Out of Scope) 3
2. SYSTEM DESIGN 4
2.1 Architecture 4
2.2 Database Design 5
2.2.1 ER Diagram (Conceptual) 5
2.1.2 ER To Table 6
2.2.3 Normalized Table (Table Definitions) 7
2.2.4 Relationship Analysis (Participation) 8
3. PL/SQL 10
3.1 Triggers 10
3.1.1 Code 10
3.1.2 Output 12
3.2 Procedure 12
3.2.1 Code 12
3.2.2 output 14
3.3 Demonstrate procedure 14
3.3.1 Code 14
3.2.2 Output 15
3.4 Driver Code 15
4. Queries 18
4.1 Drop database 18
4.1.1 code 18
4.1.2 Output 18
4.2 Create database 19
4.2.1 Code 19

4.2.2 Output: 20
4.3 Create tables and add constraints 21
4.3.1 Code 21
4.3.2 Output: 27
5. CONCLUSION 28
6. REFERENCES 29

1. INTRODUCTION

1.1 Background

Universities and educational institutions handle a vast amount of data and complex processes daily. Traditional manual methods involving paper records or scattered digital files are often inefficient, prone to errors, data redundancy, and difficulties in information retrieval and coordination. An automated, centralized management system is essential for modern institutions to operate smoothly and provide better services to students and faculty.

1.2 Problem Statement

Managing student records, course enrollments, faculty details, exam schedules, fee payments, and result processing manually is time-consuming, resource-intensive, and susceptible to errors. Key challenges include:

- Inefficiencies in student registration and course enrollment leading to potential conflicts or delays.
- Difficulties in accurately tracking student academic progress, fee dues, and payment status.
- Lack of a streamlined process for faculty to manage their courses, schedule exams efficiently, evaluate results consistently, and communicate them promptly.
- Significant administrative overhead in handling paperwork, record-keeping, approvals, rejections, and communication across departments.
- Potential data security vulnerabilities and data redundancy issues associated with decentralized or paper-based systems.

This project develops a comprehensive University Management System (UMS) to address these challenges by providing a centralized, automated, role-based, and secure platform.

1.3 Objectives

- To develop a centralized database system for efficient management of student, faculty, course, exam, fee, department, and administrative data.
- To automate core university processes, including student registration, course enrollment, exam scheduling, result evaluation and publication, and fee tracking.
- To enhance data security and ensure appropriate access levels through distinct role-based interfaces (Admin, Faculty, Student).
- To minimize manual errors and data redundancy by implementing a well-structured relational database schema with appropriate constraints.
- To provide an intuitive and user-friendly web interface for all user types to interact with the system effectively.
- To create a scalable and maintainable system architecture capable of supporting institutional needs.
- To implement automated email notifications for critical system events such as registration approvals/rejections and credential distribution.

1.4 Scope of the Project

1.4.1 Functional Scope

- **Student Management:**
    - Registration of new students with verification and approval by administrators.
    - Managing student profiles, viewing enrollment status, and handling course registrations/unenrollment.
    - Displaying personalized dashboards with exam schedules, results, and fee status.
    - Simulated fee payment functionality.
- **Faculty Management:**
    - Registration of faculty members with verification by administrators.
    - Managing faculty profiles, viewing assigned courses, and examination responsibilities.
    - Providing personalized dashboards with relevant academic information.
    - Managing exams (add, update, delete), evaluating results (mark entry, grading), and locking results.
- **Course Management:**
    - Creation, modification, and deletion of courses by administrators.
    - Management of course details (ID, name, semester, credits, price).
- **Examination Management:**
    - Scheduling exams (including type, date, duration, venue) by faculty for their assigned course.
    - Admin overview and management of exams.
    - Storing and managing exam results (marks, grades, status).
- **Fee Management:**
    - Generating fee records based on registration, course enrollment, and exams.
    - Allowing administrators to manage fee records.
    - Displaying pending/completed fee transactions on student dashboards.
    - Tracking simulated payments via Payment ID.
- **Department Management:**
    - Creating, updating, or deleting departments.
    - Appointing Heads of Departments (HODs) by administrators from active faculty.
    - Associating faculty members with relevant departments.

- **User Authentication and Authorization:**
    - Implementing role-based access control (Admin, Faculty, Student).
    - Ensuring secure login using official email (`@thapar.edu`) and password, and logout functionalities.
- **Email Notifications:**
    - Sending automated emails for registration approval/rejection, credential delivery, and account restrictions.

1.4.2. Technical Scope

- Backend Development: Using Flask (Python) as the backend framework.
- Database Management: Implementing MySQL using the PyMySQL connector for handling relational data.
- Frontend Development: Utilizing HTML, CSS, and JavaScript with Jinja2 templating for designing user interfaces.
- Email Handling: Using Flask-Mail for sending automated notifications.
- Database Initialization: Providing scripts (`database_prerequisite.py`) for database and table creation, and insertion of initial data.

1.4.3. Limitations (Out of Scope)

- The system does not include modules for Attendance Tracking, Hostel Management, or Library Management.
- Integration with external learning platforms (e.g., Moodle, Blackboard) is not covered.
- Advanced reporting, analytics, and data visualization features are not implemented.
- Real-time payment gateway integration is excluded; payments are simulated.
- SMS notifications are not implemented.

2. SYSTEM DESIGN

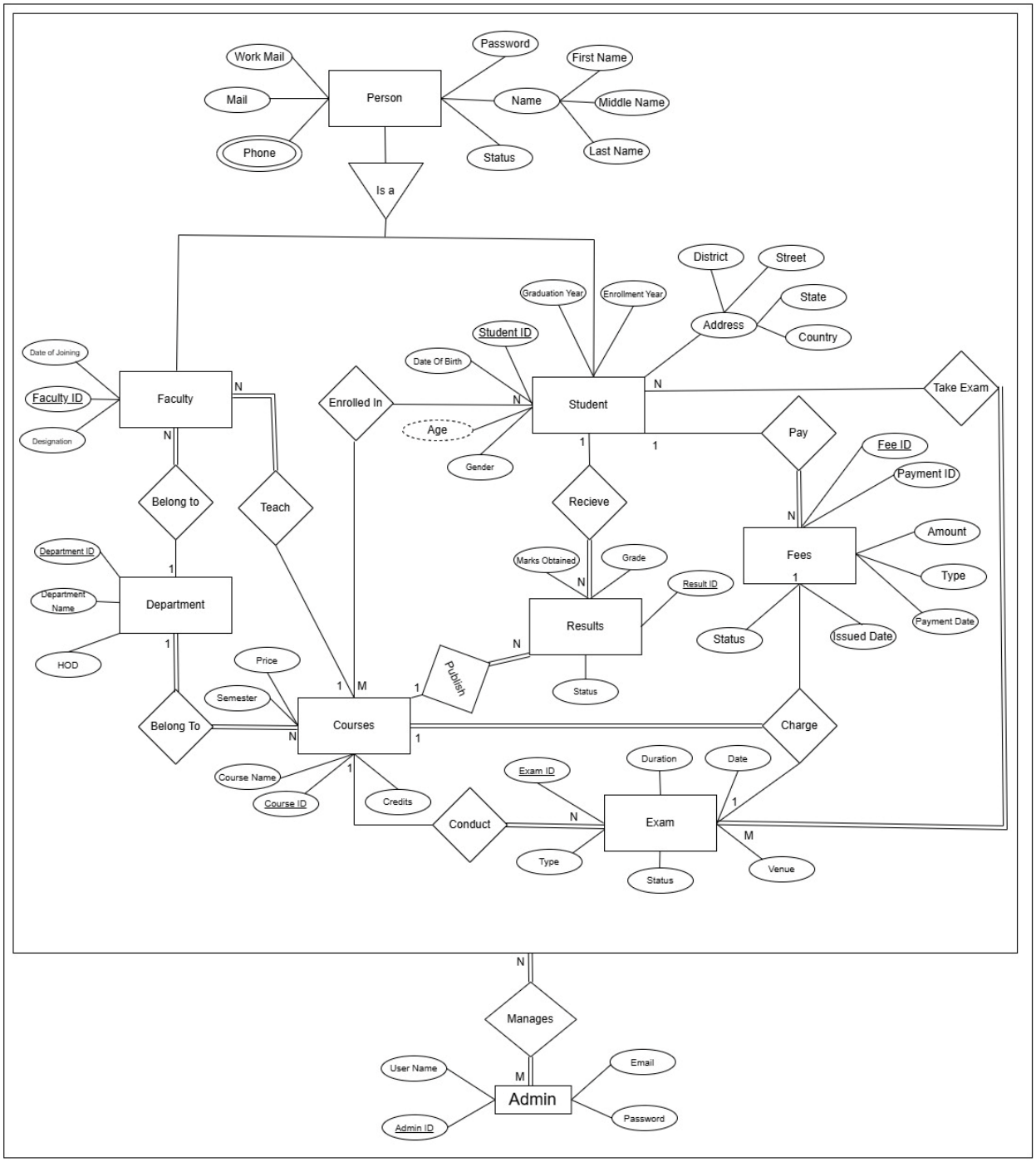
2.1 Architecture

The University Management System employs a standard **3-Tier Web Application Architecture**:

- **Presentation Tier (Client-Side)**: Web browsers render HTML pages (from the `templates/` directory) styled with CSS (`static/css/`). Basic JavaScript (`static/scripts/`) handles minor client-side interactions. Jinja2 templating dynamically generates HTML.
- **Application Tier (Server-Side)**: The Flask application (`main.py`) processes HTTP requests, manages user sessions, implements business logic (validation, grading), interacts with the database (via PyMySQL), and handles email notifications (via Flask-Mail).
- **Data Tier (Database)**: An externally hosted MySQL database (Aiven cloud) stores persistent data. The schema, defined in `database_prerequisite.py`, includes tables for students, faculty, courses, etc., with integrity enforced by constraints.

2.2 Database Design

2.2.1 ER Diagram (Conceptual)



2.1.2 ER To Table

The University Management System requires a well-structured database with 12 (including two tables for multivalued attributes of students and faculty) tables to handle various entities and their relationships. The relationships between entities are outlined below:

1. Students and Courses (N:M Relationship)

- **Tables Required:** students, courses, enrollment
- Students can enroll in multiple courses, and a course can have multiple students.
- The enrollment table establishes the relationship, with a composite primary key comprising student_id and course_id.
- An additional attribute, enrollment_date, is included.

2. Faculty and Department (1:N Relationship)

- **Tables Required:** faculty, department
- A faculty member belongs to a single department, but a department can have multiple faculty members.
- The department table is independent, while the faculty table includes department_id as a foreign key.

3. Faculty and Course (1:N Relationship)

- **Tables Required:** faculty, courses
- A faculty member can teach only one course, but a course can be taught by multiple faculty members.
- The faculty table includes course_id as a foreign key.

4. Courses and Exams (1:N Relationship)

- **Tables Required:** courses, exams
- A course can conduct multiple exams, but each exam is associated with only one course.
- The exam table includes course_id as a foreign key and exam_id as the primary key.
- A unique constraint is applied to the combination of course_id and type.

5. Courses and Results (1:N Relationship)

- **Tables Required:** courses, results
- A course can have multiple results published, but each result belongs to only one course.
- The results table includes course_id as a foreign key.

6. Students and Exams (N:M Relationship)

- **Tables Required:** students, exams, takes_exams
- A student can take multiple exams, and an exam can be given by multiple students enrolled in the same course.
- The takes_exams table contains a composite primary key of student_id and exam_id.

7. Students and Results (1:N Relationship)

- **Tables Required:** students, results
- A student can receive multiple results, but each result is associated with only one student.

- The results table includes student_id as a foreign key.

8. Students and Fees (1:N Relationship)

- **Tables Required:** students, fees
- A student can pay multiple fees, but each fee is linked to only one student.
- The fees table includes student_id as a foreign key.

9. Courses and Fees (1:1 Relationship)

- **Tables Required:** courses, fees
- Each course charges only one fee.
- The fees table includes course_id as a foreign key.

10. Exams and Fees (1:1 Relationship)

- **Tables Required:** exams, fees
- Each exam charges only one fee.
- The fees table includes exam_id as a foreign key.

11. Admin Management (1:1 Relationship)

- **Tables Required:** admin
- The admin manages all entities. Since all entities are managed by the admin, no relationships need to be explicitly established.

2.2.3 Normalized Table (Table Definitions)

The database schema comprises the following core tables (defined in `database_prerequisite.py`):

1. Student

   Students(Student_ID, First_Name, Middle_Name, Last_Name, street, district, state, country, Gender, Date_of_Birth, mail, College_Mail,Password, Enrollment_Year, Graduation_Year, Status)

2. Student Phone No

   PhoneNumbers(Student_ID, Phone)

3. Courses

   Courses(Course_ID, Course_Name, Semester, Credits, Price)

4. Enrollment

   Enrollment(Student_ID, Course_ID, Enrollment_On)

5. Fees

   Fees(Fee_ID, Student_ID,Exam_ID,Course_ID, Amount, Issued_Date, Payment_Date ,Type, Status,payment_ID)

6. Exams

Exams(Exam_ID,Course_ID, Exam_Date, Exam_Duration, Exam_Type, Venue,Status)

7. Takes Exam

Takes_Exam(Student_ID, Exam_ID,Status)

8.. Results

Results(Result_ID, Exam_ID,Student_ID, Course_ID, Marks_Obtained,Grade,Status)

9. Department

Department(Department_ID, Department_Name, Head_Of_Department)

10. Faculty

Faculty(Faculty_ID, First_Name, Middle_Name, Last_Name, Date_Of_Joining, Designation, Mail,Official_Mail,Password,Status,Course_ID,Department_ID)

11. Faculty Phone No

FacultyPhone(Faculty_ID, Phone)

12. Admin

Admin(Admin_ID, User_Name, Password, Email,Password)

2.2.4 Relationship Analysis (Participation)

The participation of entities in various relationships within the University Management System is described below:

1. **Students and Courses:**
    - **Partial Participation (Both)** A student may not be enrolled in any course, and some courses may have no students enrolled. Therefore, both entities participate partially in the relationship.
2. **Faculty and Department:**
    - **Total Participation (Faculty) & Partial Participation (Department):** Every faculty member must belong to a department, but a department can exist without any faculty members. Faculty participation is total, while department participation is partial.
3. **Faculty and Courses:**
    - **Total Participation (Faculty) & Partial Participation (Course):** Every faculty member must teach at least one course. However, some courses may not be assigned to any faculty. Therefore, faculty participation is total, while course participation is partial.

4. **Departments and Courses:**

    - **Total Participation (Both):** Every department must offer at least one course, and every course must be offered by a department. Thus, both department and course participation are total.

5. **Courses and Exams:**

    - **Total Participation (Exam) & Partial Participation (Course):** Every exam must be associated with a course, but a course may not necessarily conduct exams. Therefore, exams participate totally, while courses participate partially.

6. **Students and Exams:**

    - **Total Participation (Exam) & Partial Participation (Student):** Every exam is taken by one or more students, but some students may not have participated in any exams. Hence, exams participate totally, while students participate partially.

7. **Courses, Exams, and Fees:**

    - **Total Participation (Course & Exam) & Partial Participation (Fee):** Every course and exam have associated fees, but some fees may not be related to any specific course or exam (e.g., registration fees). Therefore, fees participate partially.

8. **Students and Fees:**

    - **Total Participation (Fee) & Partial Participation (Student):** Every fee must be associated with a student, but a student may not have paid any fees. Thus, fees participate totally, while students participate partially.

9. **Admin Management:**

    - **Total Participation (Both):** Every entity in the system is managed by an admin, and an admin cannot exist without management authority. Therefore, both participate totally.

10. **Students and Results:**

    - **Total Participation (Result) & Partial Participation (Student):** Every result is linked to a student, but not all students may have results published. Thus, results participate totally, while students participate partially.

11. **Courses and Results:**

    - **Total Participation (Result) & Partial Participation (Course):** Every result is associated with a course, but some courses may not have any results published. Therefore, results participate totally, while courses participate partially.

3. PL/SQL

3.1 Triggers

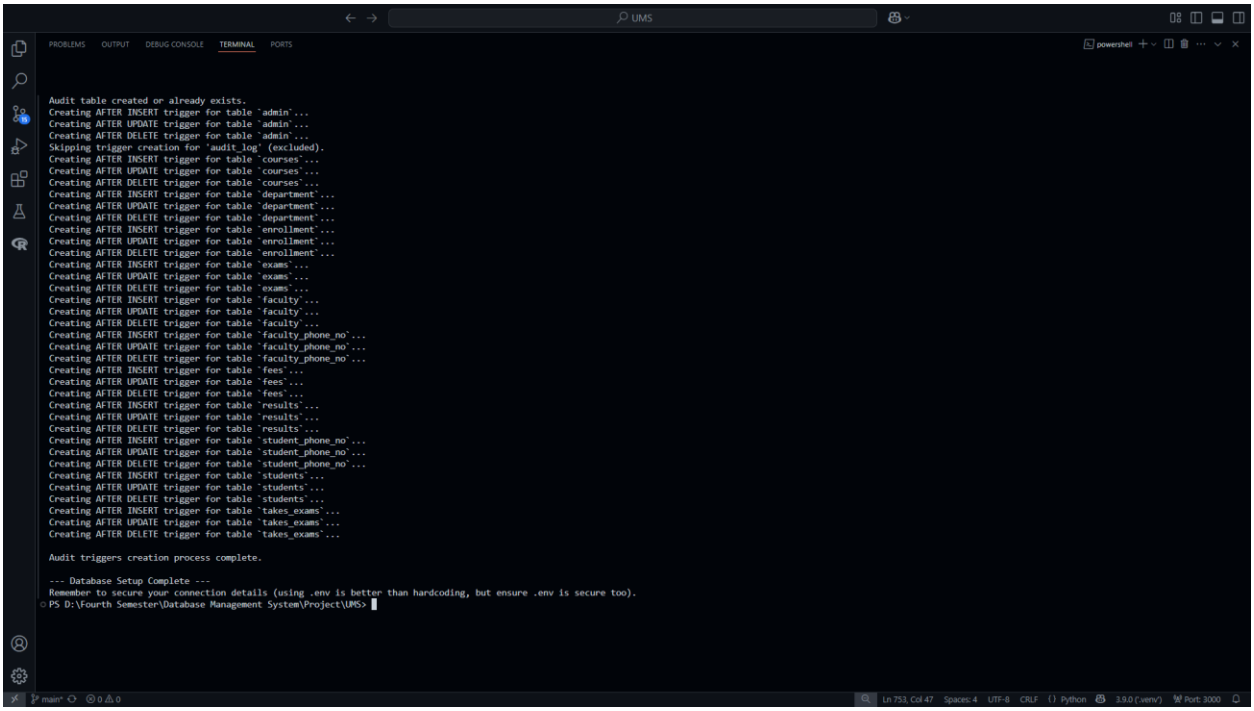
3.1.1 Code

```
def create_audit_trigger():
    connection = None
    cursor = None
    try:
        connection = get_connection(db_name=DATABASE_NAME)
        if connection is None:
            return
        cursor = connection.cursor()

        print("\nCreating audit table if it doesn't exist...")
        cursor.execute("""
        CREATE TABLE IF NOT EXISTS audit_log (
            Audit_ID INT AUTO_INCREMENT PRIMARY KEY,
            Event_Type ENUM('INSERT', 'UPDATE', 'DELETE') NOT NULL,
            Table_Name VARCHAR(255) NOT NULL,
            Event_Time DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP
        );
        """)
        print("Audit table created or already exists.")

        cursor.execute("SHOW TABLES;")
        tables = cursor.fetchall()

        excluded_tables = ['audit_log'] # IMPORTANT: Do not trigger on the audit table itself

        for table_dict in tables:
            table_name = list(table_dict.values())[0] # Get the table name

            if table_name in excluded_tables:
                print(f"Skipping trigger creation for '{table_name}' (excluded).")
                continue

            try:
                print(f"Creating AFTER INSERT trigger for table `{table_name}`...")
                cursor.execute(f"""
                CREATE TRIGGER `{table_name}_after_insert`
                AFTER INSERT ON `{table_name}`
                FOR EACH ROW
```

```
            INSERT INTO audit_log (Event_Type, Table_Name)
            VALUES ('INSERT', '{table_name}');
            """)
        except pymysql.Error as e:
            if e.args[0] == 1359:
                print(f"AFTER INSERT trigger for `{table_name}` already exists.")
            else:
                print(f"Error creating AFTER INSERT trigger for `{table_name}`: {e}",
file=sys.stderr)

        try:
            print(f"Creating AFTER UPDATE trigger for table `{table_name}`...")
            cursor.execute(f"""
            CREATE TRIGGER `{table_name}_after_update`
            AFTER UPDATE ON `{table_name}`
            FOR EACH ROW
            INSERT INTO audit_log (Event_Type, Table_Name)
            VALUES ('UPDATE', '{table_name}');
            """)
        except pymysql.Error as e:
            if e.args[0] == 1359: # Error code 1359 is "Trigger already exists"
                print(f"AFTER UPDATE trigger for `{table_name}` already exists.")
            else:
                print(f"Error creating AFTER UPDATE trigger for `{table_name}`: {e}",
file=sys.stderr)

        try:
            print(f"Creating AFTER DELETE trigger for table `{table_name}`...")
            cursor.execute(f"""
            CREATE TRIGGER `{table_name}_after_delete`
            AFTER DELETE ON `{table_name}`
            FOR EACH ROW
            INSERT INTO audit_log (Event_Type, Table_Name)
            VALUES ('DELETE', '{table_name}');
            """)
        except pymysql.Error as e:
            if e.args[0] == 1359:
                print(f"AFTER DELETE trigger for `{table_name}` already exists.")
            else:
                print(f"Error creating AFTER DELETE trigger for `{table_name}`: {e}",
file=sys.stderr)

    connection.commit()
    print("\nAudit triggers creation process complete.")
```

```
    except pymysql.Error as err:
        print(f"Error during audit trigger setup: {err}", file=sys.stderr)
        if connection:
            connection.rollback()
    finally:
        if cursor:
            cursor.close()
        if connection:
            connection.close()
```

3.1.2 Output



3.2 Procedure

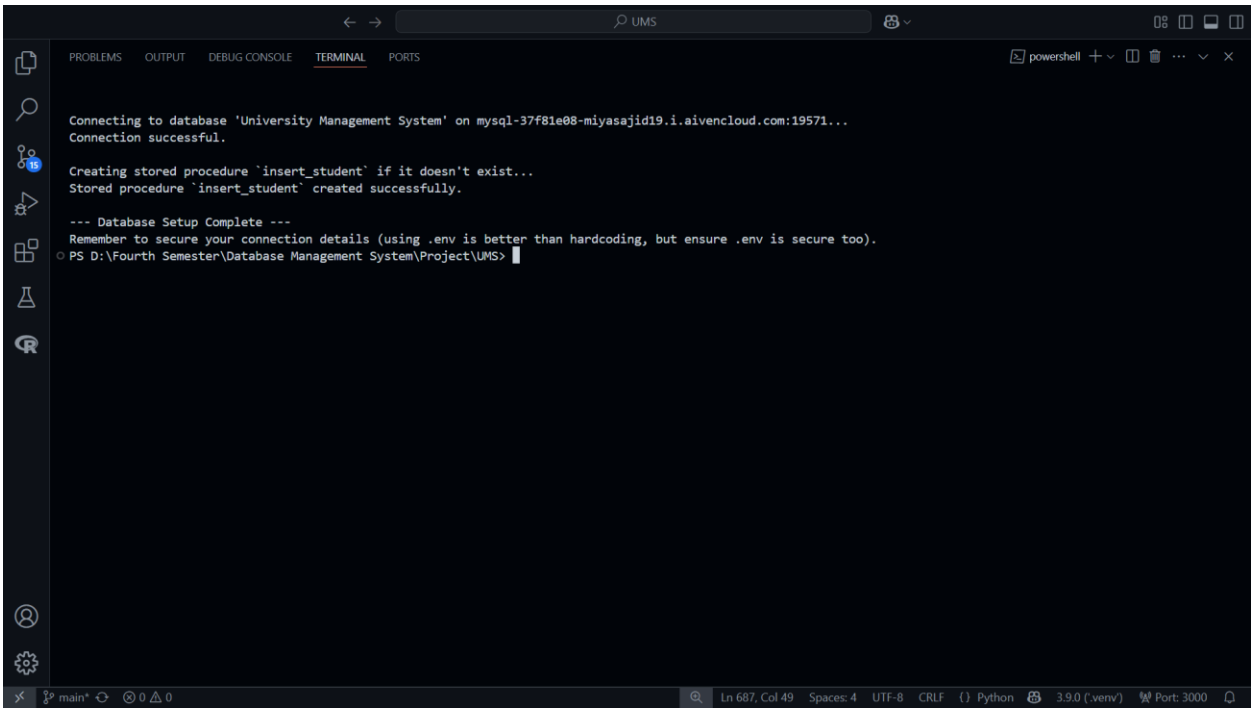
3.2.1 Code

```
def create_procedure():
    connection = None
    cursor = None
    try:
        connection = get_connection(db_name=DATABASE_NAME)
        if connection is None:
            return
        cursor = connection.cursor()
        print("\nCreating stored procedure `insert_student` if it doesn't exist...")
```

```
        cursor.execute("""
        CREATE PROCEDURE insert_student (
            IN p_First_Name VARCHAR(255),
            IN p_Middle_Name VARCHAR(255),
            IN p_Last_Name VARCHAR(255),
            IN p_Street VARCHAR(255),
            IN p_District VARCHAR(255),
            IN p_State VARCHAR(255),
            IN p_Country VARCHAR(255),
            IN p_Gender ENUM('Male', 'Female', 'Others'),
            IN p_Date_of_Birth DATE,
            IN p_Email VARCHAR(255),
            IN p_College_Email VARCHAR(255),
            IN p_Password VARCHAR(255),
            IN p_Enrollment_Year YEAR
        )
        BEGIN
            DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
            BEGIN
                ROLLBACK;
                SELECT 'Error: Unable to insert student. Please check the input data.' AS Error;
            END;

            START TRANSACTION;

            INSERT INTO students (
                First_Name, Middle_Name, Last_Name, Street, District, State,
                Country, Gender, Date_of_Birth, Email, College_Email, Password, Enrollment_Year
            ) VALUES (
                p_First_Name, p_Middle_Name, p_Last_Name, p_Street, p_District, p_State,
                p_Country, p_Gender, p_Date_of_Birth, p_Email, p_College_Email, p_Password,
p_Enrollment_Year
            );

            COMMIT;

            SELECT 'Student inserted successfully.' AS Message;
        END;
        """)
        print("Stored procedure `insert_student` created successfully.")
    except pymysql.Error as err:
        print(f"Error creating stored procedure: {err}", file=sys.stderr)
    finally:
        if cursor:
```

```
        cursor.close()
    if connection:
        connection.close()
```

3.2.2 output



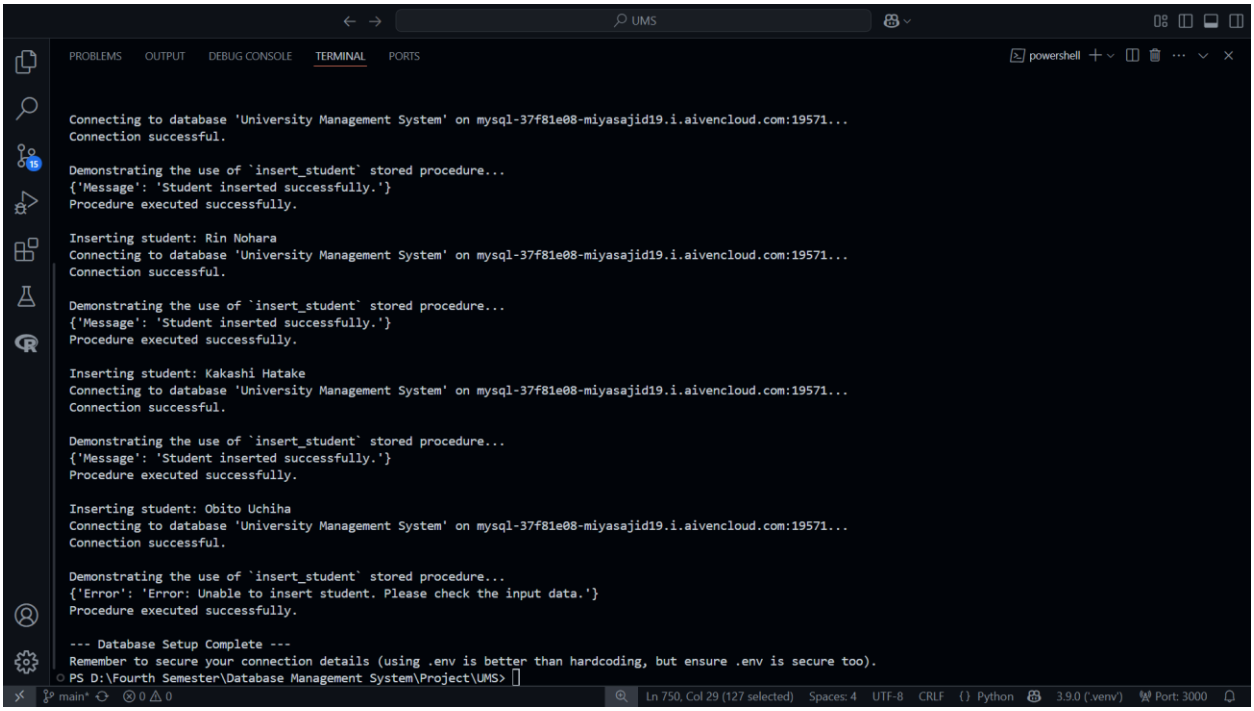
3.3 Demonstrate procedure

3.3.1 Code

```
def demonstrate_procedure(student_data):
    connection = None
    cursor = None
    try:
        connection = get_connection(db_name=DATABASE_NAME)
        if connection is None:
            return
        cursor = connection.cursor()
        print("\nDemonstrating the use of `insert_student` stored procedure...")
        cursor.callproc('insert_student', (
            student_data['First_Name'], student_data.get('Middle_Name', ''),
student_data['Last_Name'],
            student_data['Street'], student_data['District'], student_data['State'],
            student_data['Country'], student_data['Gender'], student_data['Date_of_Birth'],
            student_data['Email'], student_data['College_Email'], student_data['Password'],
            student_data['Enrollment_Year']
        ))
```

```
        result = cursor.fetchall()
        for row in result:
            print(row)
        connection.commit()
        print("Procedure executed successfully.")
    except pymysql.Error as err:
        print(f"Error demonstrating procedure: {err}", file=sys.stderr)
    finally:
        if cursor:
            cursor.close()
        if connection:
            connection.close()
```

3.2.2 Output



3.4 Driver Code

```
if __name__ == "__main__":
    create_audit_trigger()
    create_procedure()
    students = [
    {
        'First_Name': 'Obito',
        'Middle_Name': None,
        'Last_Name': 'Uchiha',
```

```
    'Street': 'Hidden Leaf Village',
    'District': 'Konoha',
    'State': 'Land of Fire',
    'Country': 'Japan',
    'Gender': 'Male',
    'Date_of_Birth': '1990-02-10',
    'Email': 'obito.uchiha@example.com',
    'College_Email': 'obito@thapar.edu',
    'Password': 'Mangekyo123',
    'Enrollment_Year': 2010
},
{
    'First_Name': 'Rin',
    'Middle_Name': None,
    'Last_Name': 'Nohara',
    'Street': 'Hidden Leaf Village',
    'District': 'Konoha',
    'State': 'Land of Fire',
    'Country': 'Japan',
    'Gender': 'Female',
    'Date_of_Birth': '1992-06-15',
    'Email': 'rin.nohara@example.com',
    'College_Email': 'rin@thapar.edu',
    'Password': 'HealingJutsu456',
    'Enrollment_Year': 2011
},
{
    'First_Name': 'Kakashi',
    'Middle_Name': None,
    'Last_Name': 'Hatake',
    'Street': 'Hidden Leaf Village',
    'District': 'Konoha',
    'State': 'Land of Fire',
    'Country': 'Japan',
    'Gender': 'Male',
    'Date_of_Birth': '1989-09-15',
    'Email': 'kakashi.hatake@example.com',
    'College_Email': 'kakashi@thapar.edu',
    'Password': 'Sharingan789',
    'Enrollment_Year': 2009
},{
    'First_Name': 'Obito',
    'Middle_Name': None,
    'Last_Name': 'Uchiha',
```

```
    'Street': 'Hidden Leaf Village',
    'District': 'Konoha',
    'State': 'Land of Fire',
    'Country': 'Japan',
    'Gender': 'Male',
    'Date_of_Birth': '1990-02-10',
    'Email': 'obito.uchiha@example.com',
    'College_Email': 'obito@thapar.edu',
    'Password': 'Mangekyo123',
    'Enrollment_Year': 2010
}
]
for student in students:
    print(f"\nInserting student: {student['First_Name']} {student['Last_Name']}")
    demonstrate_procedure(student)
```

4. Queries

These are some queries used in this project :

4.1 Drop database

4.1.1 code

```
def drop_database():
    """Drops the specified database if it exists."""
    connection = None
    cursor = None
    try:
        # Connect without specifying a database initially to run DROP DATABASE
        connection = get_connection()
        if connection is None:
            return # Connection failed, already printed error

        cursor = connection.cursor()
        # Use backticks around the database name in SQL
        print(f"\nDropping database `{DATABASE_NAME}` if it exists...")
        cursor.execute(f"DROP DATABASE IF EXISTS `{DATABASE_NAME}`;")
        connection.commit()
        print(f"Database `{DATABASE_NAME}` dropped successfully.")
    except pymysql.Error as err:
        print(f"Error dropping database: {err}", file=sys.stderr)
    finally:
        if cursor:
            cursor.close()
        if connection:
            connection.close()
```

4.1.2 Output

```
PS D:\Fourth Semester\Database Management System\Project\UMS> py pl_sql.py
--- University Management System Database Setup ---
Connecting to database 'default' on mysql-37f81e08-miyasajid19.i.aivencloud.com:19571...
Connection successful.

Dropping database `University Management System` if it exists...
Database `University Management System` dropped successfully.

--- Database Setup Complete ---
Remember to secure your connection details (using .env is better than hardcoding, but ensure .env is secure too).
PS D:\Fourth Semester\Database Management System\Project\UMS>
```

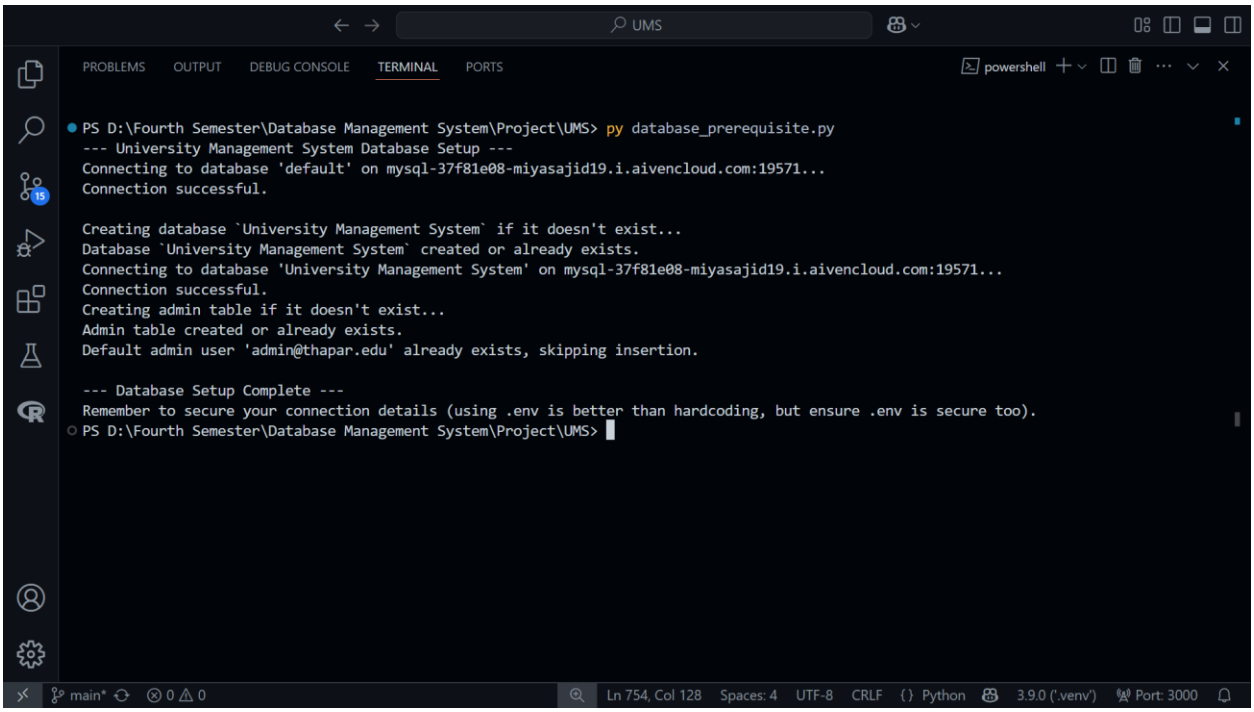
4.2 Create database

4.2.1 Code

```
def create_database():
    connection = None
    cursor = None
    try:
        connection = get_connection()
        if connection is None:
            return

        cursor = connection.cursor()
        print(f"\nCreating database `{DATABASE_NAME}` if it doesn't exist...")
        cursor.execute(f"CREATE DATABASE IF NOT EXISTS `{DATABASE_NAME}`;")
        print(f"Database `{DATABASE_NAME}` created or already exists.")

        if cursor: cursor.close()
        if connection: connection.close()

        # Now connect to the specific database to create tables within it
        connection = get_connection(db_name=DATABASE_NAME)
        if connection is None:
            # Connection to specific DB failed, exit setup
            print("Could not connect to the newly created database. Exiting.", file=sys.stderr)
            sys.exit(1)

        cursor = connection.cursor()

        print("Creating admin table if it doesn't exist...")
        cursor.execute("""
        CREATE TABLE IF NOT EXISTS admin (
            Admin_ID INT PRIMARY KEY AUTO_INCREMENT,
            User_Name VARCHAR(255) NOT NULL,
            Email VARCHAR(255) NOT NULL UNIQUE,
            Password VARCHAR(255) NOT NULL
        );
        """)
        print("Admin table created or already exists.")

        cursor.execute("SELECT COUNT(*) FROM admin WHERE Email = 'admin@thapar.edu'")
        count = cursor.fetchone()['COUNT(*)']

        if count == 0:
            print("Inserting initial default admin data...")
```

```
            cursor.execute("""
            INSERT INTO admin (User_Name, Email, Password) VALUES
            ('admin', 'admin@thapar.edu', 'admin@tiet');
            """)
            print("Initial default admin data inserted.")
            connection.commit()
        else:
            print("Default admin user 'admin@thapar.edu' already exists, skipping insertion.")

    except pymysql.Error as err:
        print(f"Error creating database or admin table: {err}", file=sys.stderr)
        if connection:
            connection.rollback()
    finally:
        if cursor:
            cursor.close()
        if connection:
            connection.close()
```

4.2.2 Output:



4.3 Create tables and add constraints

4.3.1 Code

```
def create_tables():
    connection = None
    cursor = None
    try:
        connection = get_connection(db_name=DATABASE_NAME)
        if connection is None:
            return

        cursor = connection.cursor()
        print(f"\nCreating tables in database `{DATABASE_NAME}`...")

        tables_sql = [
            ("admin", """
                CREATE TABLE IF NOT EXISTS admin (
                    Admin_ID INT PRIMARY KEY AUTO_INCREMENT,
                    User_Name VARCHAR(255) NOT NULL,
                    Email VARCHAR(255) NOT NULL UNIQUE,
                    Password VARCHAR(255) NOT NULL
                );
            """),
            ("students", """
                CREATE TABLE IF NOT EXISTS students (
                    Student_ID INT AUTO_INCREMENT PRIMARY KEY,
                    First_Name VARCHAR(255) NOT NULL,
                    Middle_Name VARCHAR(255),
                    Last_Name VARCHAR(255) NOT NULL,
                    Street VARCHAR(255) NOT NULL,
                    District VARCHAR(255) NOT NULL,
                    State VARCHAR(255) NOT NULL DEFAULT 'Nepal',
                    Country VARCHAR(255) NOT NULL,
                    Gender ENUM('Male', 'Female', 'Others') NOT NULL,
                    Date_of_Birth DATE NOT NULL,
                    Email VARCHAR(255) UNIQUE NOT NULL CHECK (Email LIKE '%@%'),
                    College_Email VARCHAR(255) UNIQUE NOT NULL CHECK (College_Email LIKE
'%@thapar.edu'),
                    Password VARCHAR(255) NOT NULL CHECK (LENGTH(Password) >= 8),
                    Enrollment_Year YEAR NOT NULL,
                    Graduation_Year YEAR NOT NULL DEFAULT (Enrollment_Year + 4),
                    Status ENUM('Pending', 'Enrolled', 'Graduated', 'Restricted') NOT NULL DEFAULT
'Pending'
                );
```

```
        """),
        ("courses", """
            CREATE TABLE IF NOT EXISTS courses (
                Course_ID VARCHAR(13) PRIMARY KEY,
                Course_Name VARCHAR(55) NOT NULL UNIQUE,
                Semester ENUM('1', '2', '3', '4', '5', '6', '7', '8') NOT NULL,
                Credits FLOAT NOT NULL,
                Price FLOAT NOT NULL DEFAULT 0
            );
        """),
         ("department", """
            CREATE TABLE IF NOT EXISTS department (
                Department_ID VARCHAR(7) PRIMARY KEY,
                Department_Name VARCHAR(255) NOT NULL UNIQUE,
                Head_of_Department INT -- This FK will be added later as it references 'faculty'
            );
        """),
        ("faculty", """
            CREATE TABLE IF NOT EXISTS faculty (
                Faculty_ID INT PRIMARY KEY AUTO_INCREMENT,
                First_Name VARCHAR(255) NOT NULL,
                Middle_Name VARCHAR(255),
                Last_Name VARCHAR(255) NOT NULL,
                Date_of_Joining DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
                Designation VARCHAR(32) NOT NULL,
                Mail VARCHAR(255) NOT NULL, # Assuming this is a personal mail
                Official_Mail VARCHAR(255) UNIQUE NOT NULL, # This should be unique like
student email
                Password VARCHAR(255) NOT NULL,
                Course_ID VARCHAR(13), # Changed to SET NULL
                Department_ID VARCHAR(7) NOT NULL,
                Status ENUM('Pending', 'Active') DEFAULT 'Pending',
                FOREIGN KEY (Course_ID) REFERENCES courses(Course_ID) ON DELETE
SET NULL ON UPDATE CASCADE, # Changed to SET NULL
                FOREIGN KEY (Department_ID) REFERENCES department(Department_ID) ON
DELETE CASCADE ON UPDATE CASCADE
            );
        """),
        ("student_phone_no", """
            CREATE TABLE IF NOT EXISTS student_phone_no (
                Student_ID INT NOT NULL,
                Phone VARCHAR(14) NOT NULL,
                CONSTRAINT PRIMARY KEY (Student_ID, Phone),
```

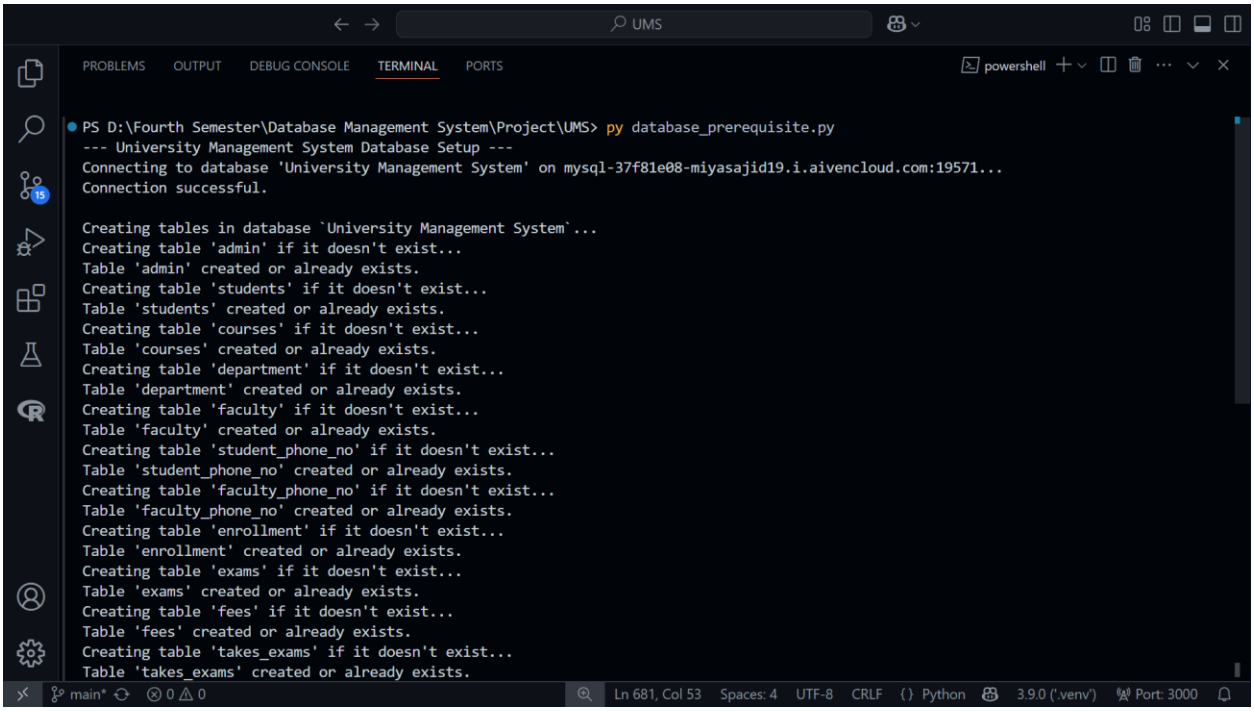
```
            CONSTRAINT FK_Student FOREIGN KEY (Student_ID) REFERENCES
students(Student_ID) ON DELETE CASCADE ON UPDATE CASCADE
        );
    """),
    ("faculty_phone_no", """
        CREATE TABLE IF NOT EXISTS faculty_phone_no (
            Faculty_ID INT NOT NULL,
            Phone VARCHAR(14) NOT NULL,
            CONSTRAINT PRIMARY KEY (Faculty_ID, Phone),
            CONSTRAINT FK_Faculty_Phone FOREIGN KEY (Faculty_ID) REFERENCES
faculty(Faculty_ID) ON DELETE CASCADE ON UPDATE CASCADE
        );
    """),
    ("enrollment", """
        CREATE TABLE IF NOT EXISTS enrollment (
            Student_ID INT NOT NULL,
            Course_ID VARCHAR(13) NOT NULL,
            Enrolled_IN DATE DEFAULT (CURRENT_DATE), -- Added default date
            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,
            PRIMARY KEY (Student_ID, Course_ID)
        );
    """),
    ("exams", """
        CREATE TABLE IF NOT EXISTS exams (
            Exam_ID INT PRIMARY KEY AUTO_INCREMENT,
            Course_ID VARCHAR(13),
            Exam_Date DATE NOT NULL,
            Exam_Duration FLOAT NOT NULL,
            Exam_Type ENUM ('Mid Semester Test', 'End Semester Test', 'Quiz-1', 'Quiz-2',
'Lab Evaluation I', 'Lab Evaluation II', 'Others') NOT NULL,
            Venue VARCHAR(61) NOT NULL,
            Status ENUM('Unevaluated','Evaluated','Locked') NOT NULL,
            CONSTRAINT UNIQUE(Course_ID, Exam_Type), # Ensure unique exam type per
course
            FOREIGN KEY (Course_ID) REFERENCES courses(Course_ID) ON DELETE
CASCADE ON UPDATE CASCADE
        );
    """),
     ("fees", """
        CREATE TABLE IF NOT EXISTS fees (
            Fee_ID INT PRIMARY KEY AUTO_INCREMENT,
```

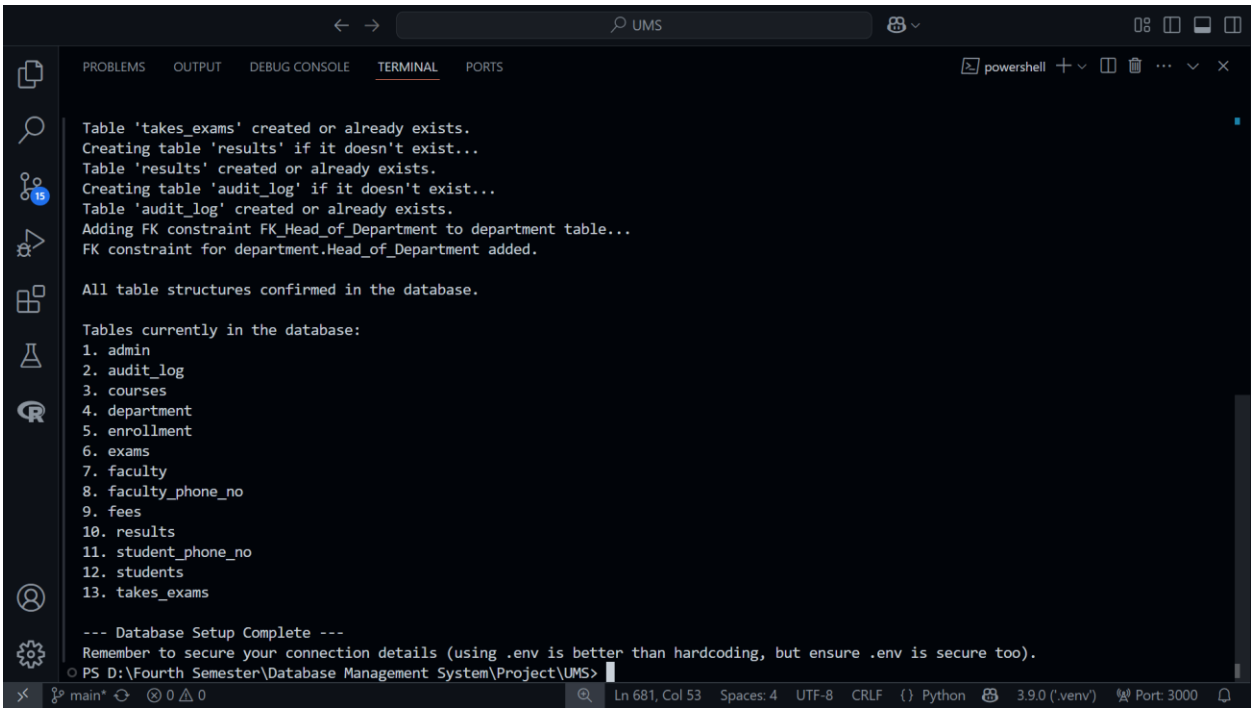
```
            Student_ID INT,
            Exam_ID INT, -- Fee could be exam related
            Course_ID VARCHAR(13), -- Fee could be course registration related
            Amount FLOAT NOT NULL,
            Issued_Date DATE NOT NULL DEFAULT (CURRENT_DATE),
            Type ENUM('Registration Fees','Course Registration','Exam Fee','Other') NOT
NULL, # Added 'Other'
            Payment_Date DATE,
            Status ENUM('Pending','Paid') DEFAULT 'Pending',
            Payment_ID VARCHAR(13), -- Might need to be UNIQUE depending on
requirements
            FOREIGN KEY (Student_ID) REFERENCES students(Student_ID) ON DELETE
CASCADE ON UPDATE CASCADE,
            FOREIGN KEY (Exam_ID) REFERENCES exams(Exam_ID) ON DELETE
CASCADE ON UPDATE CASCADE,
            FOREIGN KEY (Course_ID) REFERENCES courses(Course_ID) ON DELETE
CASCADE ON UPDATE CASCADE

        );
    """),
    ("takes_exams", """
        CREATE TABLE IF NOT EXISTS takes_exams (
            Student_ID INT NOT NULL,
            Exam_ID INT NOT NULL,
            Status ENUM('Unevaluated','Evaluated','Locked') DEFAULT 'Unevaluated',
            FOREIGN KEY (Student_ID) REFERENCES students(Student_ID) ON DELETE
CASCADE ON UPDATE CASCADE,
            FOREIGN KEY (Exam_ID) REFERENCES exams(Exam_ID) ON DELETE
CASCADE ON UPDATE CASCADE,
            PRIMARY KEY (Student_ID, Exam_ID)
        );
    """),
     ("results", """
        CREATE TABLE IF NOT EXISTS results (
            Result_ID INT PRIMARY KEY AUTO_INCREMENT,
            Exam_ID INT NOT NULL,
            Student_ID INT NOT NULL,
            Course_ID VARCHAR(13) NOT NULL, -- Result is for a specific exam *in* a
specific course
            Marks_Obtained FLOAT,
            Grade ENUM('A', 'A-', 'B', 'B-', 'C', 'C-', 'D', 'E', 'F'), -- Added F for failing
            Status ENUM('Unevaluated','Evaluated','Locked') DEFAULT 'Unevaluated',
            FOREIGN KEY (Exam_ID) REFERENCES exams(Exam_ID) ON DELETE
CASCADE ON UPDATE CASCADE,
```

```
            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,
            -- Ensure the student is enrolled in the course related to the exam? This is complex
and often handled by application logic.
            -- For now, just ensure unique entry per exam/student/course combo.
            UNIQUE (Exam_ID, Student_ID, Course_ID)
        );
    """),
    ("audit_log", """
        CREATE TABLE IF NOT EXISTS audit_log (
            Audit_ID INT AUTO_INCREMENT PRIMARY KEY,
            Event_Type ENUM('INSERT', 'UPDATE', 'DELETE') NOT NULL,
            Table_Name VARCHAR(255) NOT NULL,
            Event_Time DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP
        );
    """)
]

for table_name, sql in tables_sql:
    print(f"Creating table '{table_name}' if it doesn't exist...")
    cursor.execute(sql)
    print(f"Table '{table_name}' created or already exists.")

print("Adding FK constraint FK_Head_of_Department to department table...")
try:
    cursor.execute(f"""
        SELECT COUNT(*)
        FROM information_schema.TABLE_CONSTRAINTS
        WHERE CONSTRAINT_SCHEMA = '{DATABASE_NAME}'
        AND TABLE_NAME = 'department'
        AND CONSTRAINT_NAME = 'FK_Head_of_Department'
        AND CONSTRAINT_TYPE = 'FOREIGN KEY';
    """)
    constraint_exists = cursor.fetchone()['COUNT(*)'] > 0

    if not constraint_exists:
        cursor.execute("""
        ALTER TABLE department
        ADD CONSTRAINT FK_Head_of_Department
        FOREIGN KEY (Head_of_Department) REFERENCES faculty(Faculty_ID)
        ON DELETE SET NULL -- Use SET NULL because the HOD might be removed
        ON UPDATE CASCADE;
```

```
            """)
            print("FK constraint for department.Head_of_Department added.")
        else:
            print("FK constraint for department.Head_of_Department already exists.")

    except pymysql.Error as e:
        print(f"Error adding FK constraint to department table: {e}", file=sys.stderr)

    connection.commit()
    print("\nAll table structures confirmed in the database.")

    print("\nTables currently in the database:")
    cursor.execute("SHOW TABLES;")
    tables = cursor.fetchall()
    if tables:
        for index, table_dict in enumerate(tables):
            table_name = list(table_dict.values())[0]
            print(f"{index+1}. {table_name}")
    else:
        print("No tables found in the database.")

except pymysql.Error as err:
    print(f"Error creating tables or constraints: {err}", file=sys.stderr)
    if connection:
        connection.rollback()
finally:
    if cursor:
        cursor.close()
    if connection:
        connection.close()
```

4.3.2 Output:


UMS
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
powershell
PS D:\Fourth Semester\Database Management System\Project\UMS> py database_prerequisite.py
--- University Management System Database Setup ---
Connecting to database 'University Management System' on mysql-37f81e08-miyasajid19.i.aivencloud.com:19571...
Connection successful.
Creating tables in database `University Management System`...
Creating table 'admin' if it doesn't exist...
Table 'admin' created or already exists.
Creating table 'students' if it doesn't exist...
Table 'students' created or already exists.
Creating table 'courses' if it doesn't exist...
Table 'courses' created or already exists.
Creating table 'department' if it doesn't exist...
Table 'department' created or already exists.
Creating table 'faculty' if it doesn't exist...
Table 'faculty' created or already exists.
Creating table 'student_phone_no' if it doesn't exist...
Table 'student_phone_no' created or already exists.
Creating table 'faculty_phone_no' if it doesn't exist...
Table 'faculty_phone_no' created or already exists.
Creating table 'enrollment' if it doesn't exist...
Table 'enrollment' created or already exists.
Creating table 'exams' if it doesn't exist...
Table 'exams' created or already exists.
Creating table 'fees' if it doesn't exist...
Table 'fees' created or already exists.
Creating table 'takes_exams' if it doesn't exist...
Table 'takes_exams' created or already exists.
main*
Ln 681, Col 53
Spaces: 4
UTF-8
CRLF
Python
3.9.0 ('.venv')
Port: 3000


UMS
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
powershell
Table 'takes_exams' created or already exists.
Creating table 'results' if it doesn't exist...
Table 'results' created or already exists.
Creating table 'audit_log' if it doesn't exist...
Table 'audit_log' created or already exists.
Adding FK constraint FK_Head_of_Department to department table...
FK constraint for department.Head_of_Department added.
All table structures confirmed in the database.
Tables currently in the database:
1. admin
2. audit_log
3. courses
4. department
5. enrollment
6. exams
7. faculty
8. faculty_phone_no
9. fees
10. results
11. student_phone_no
12. students
13. takes_exams
--- Database Setup Complete ---
Remember to secure your connection details (using .env is better than hardcoding, but ensure .env is secure too).
PS D:\Fourth Semester\Database Management System\Project\UMS>
main*
Ln 681, Col 53
Spaces: 4
UTF-8
CRLF
Python
3.9.0 ('.venv')
Port: 3000

5. CONCLUSION

The University Management System project successfully delivers a comprehensive web application tailored for managing essential academic and administrative functions at Thapar Institute. By utilizing the Flask framework, a MySQL database, and standard web technologies, the system provides a centralized, efficient, and role-based platform for students, faculty, and administrators. It effectively automates critical processes such as registration, enrollment, exam management, fee tracking, and result processing, thereby reducing manual workload and the potential for errors. The inclusion of features like automated email notifications, dynamic credential generation, and percentile-based grading enhances its utility. The project demonstrates a practical application of database design principles and full-stack web development to address real-world challenges in university administration, providing a solid foundation for future expansion and improvement.

6. REFERENCES

1. Flask Documentation: https://flask.palletsprojects.com/
2. PyMySQL Documentation: https://pymysql.readthedocs.io/
3. MySQL Documentation: https://dev.mysql.com/doc/
4. Jinja2 Documentation: https://jinja.palletsprojects.com/
5. Flask-Mail Documentation: https://pythonhosted.org/Flask-Mail/
6. HTML/CSS/JavaScript References: MDN Web Docs (https://developer.mozilla.org/)
7. Python `datetime` Module: https://docs.python.org/3/library/datetime.html
8. Python `re` Module: https://docs.python.org/3/library/re.html
9. Aiven (Database Hosting): https://aiven.io/
10. Render (Deployment Platform): https://render.com/
11. GitHub Repository: https://github.com/miyasajid19/University-Management-System.git
12. Live Application URL: https://university-management-system-xvfp.onrender.com