D.Y. PATIL COLLEGE OF ENGINEERING, AKURDI-44, PUNE

DEPARTMENT OF

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

DBMS Mini Project

" Student Management System "

Submitted by:

Roll no	**Student Name**
TEAIDA42	Harshal Bandgar
TEAIDA53	Tejas Kulkarni
TEAIDA55	Ruturaj Mahekar
TEAIDA56	Parth Wagholikar

Under the guidance of

Ms. R.T. Ingle

Academic Year

2024-2025 (SEM I)

1. Introduction

1.1 Overview

The **Student Management System** is a web-based application designed to efficiently manage student records such as roll numbers, names, and marks. Using the **Flask framework** for backend development and **MySQL** as the database, this system provides basic CRUD (Create, Read, Update, Delete) operations. With an intuitive user interface, the system allows easy insertion, deletion, and updating of student information, and displays individual or all student records.

This project demonstrates how to integrate a Python-based web application with a relational database, providing hands-on experience in **web development** and **database management**.

1.2 Objective

The main objectives of the project are:

1. To develop a **simple, user-friendly web application** that can handle student data effectively.
2. To provide **CRUD functionality** (Insert, Delete, Update, Show) for managing student records.
3. To integrate **Flask and MySQL** for seamless interaction between the web interface and the database.
4. To demonstrate the importance of using **primary keys** for data integrity and consistency.

5. To allow **real-time operations** such as searching and displaying individual or all student records

1.3 Problem Statement

Managing student data manually can be difficult and prone to human error, especially in environments with a large number of students. Traditional systems like pen-and-paper registers or simple Excel sheets are inefficient, lack scalability, and make it difficult to perform data updates, deletions, or retrievals in real-time.

The lack of a centralized, user-friendly system makes it challenging to ensure data accuracy and quick access to student information. This project aims to address these issues by building a **web-based student management system** that simplifies the process of maintaining, updating, and retrieving student records efficiently.

1.4 Project Scope

- **CRUD Operations**:
    - Users can **insert new student data**, **update existing records**, **delete records** by roll number, and **view individual or all student data**.
- **Database Integration**:
    - Use **MySQL** for storing and retrieving student information with **Roll Number** as the primary key to ensure unique identification.
- **User Interaction**:
    - A simple web interface powered by **Flask** will be provided for easy interaction with the backend.
- **Error Handling and Validation**:
    - Ensure proper validation for data input (e.g., no duplicate roll numbers, marks within a valid range) and handle common errors gracefully.
- **Scalability**:
    - The system can be extended to include additional features such as filtering students by grades, generating reports, or adding user authentication.
- **Platform**:
    - The project will run on any device that supports a web browser (PC, laptop, etc.), making it accessible without platform dependency.

2. Methodology

The **methodology** for the **Student Management System Using Flask and MySQL** involves a structured approach that integrates **software development, database design, and testing**. The following steps outline the methods and tools used throughout the project:

Step 1: Requirement Analysis

- Identify the functional requirements:
    - The system should allow **CRUD operations**: Insert, Update, Delete, and Display records.
    - Roll Number must be treated as the **primary key** to ensure uniqueness.
    - User input should be validated to avoid invalid entries.
- Determine non-functional requirements:
    - The system must be **user-friendly**, lightweight, and responsive.

Step 2: Technology Stack Selection

- **Backend**: Flask (Python micro-framework) for handling requests and implementing business logic.
- **Database**: MySQL for persistent data storage.
- **Frontend**: HTML/CSS for basic web interface and forms to interact with the system.
- **Tools**:
    - **MySQL Workbench**: For database management and testing.
    - **Git Bash/Terminal**: To run the Flask app and execute commands.

Step 3: System Design

1. **Database Design:**
    - Create a **students** table in MySQL with the following schema:

students (RollNo INT PRIMARY KEY, Name VARCHAR(100), Marks INT)

    - Establish **relationships** (if needed) and ensure data consistency.

2. **Application Flow Design:**
    - Define **routes** for each operation (Insert, Delete, Update, Display) using Flask:
        - **/insert**: Accepts student data via form input.
        - **/update**: Takes RollNo and new values for an update.
        - **/delete**: Deletes a student by RollNo.
        - **/show**: Displays all or specific student records.

Step 4: Implementation

1. **Set up Flask**:
    - Install Flask using pip install flask.
    - Create Flask routes for different operations (Insert, Update, Delete, and Show).
2. **Database Connection**:
    - Use mysql-connector-python to connect Flask with MySQL.
    - Perform queries within Flask routes to interact with the MySQL database.
3. **User Interface**:
    - Build simple HTML forms to accept input for Insert, Update, and Delete operations.
    - Display records using tables on the webpage.

Step 5: Testing and Validation

- **Unit Testing**: Test each route (Insert, Update, Delete, Show) to ensure they work as expected.
- **Validation**:
    - Check for duplicate Roll Numbers.
    - Validate marks to ensure they are within a specific range.

- **Error Handling**: Handle potential errors such as invalid inputs or database connection failures.

Step 6: Deployment

- **Local Deployment**:
    - Run the app locally using:

      ```
      python app.py
      ```
    - Access the web interface on **http://127.0.0.1:5000/**.

Step 7: Maintenance and Future Enhancements

- Monitor the system for bugs or errors.
- Future improvements could include:
    - **User Authentication**: Add login functionality.
    - **Data Analysis and Reports**: Generate student performance reports.
    - **Advanced Search**: Allow filtering students by marks or names.

This methodology ensures the system is **efficiently planned, implemented, tested, and maintained**, with room for further enhancements.

---

3. Implementation of Student Management System Using Flask and MySQL

The implementation follows a structured workflow that involves designing the database, building the backend, creating a user interface, and testing the complete system.

1. Set up the Environment

- Install necessary tools:
    - **Python** (for Flask backend development).
    - **MySQL** (for the database).
    - **MySQL Workbench** (for managing the database).
    - Install libraries like Flask and MySQL connector using pip.
- Create the project directory and required files (HTML templates and app.py).

2. Database Design

- Create a **MySQL database** named `student_db`.
- Define a table students with three attributes:
    1. **RollNo**: Primary key (unique identifier for each student).
    2. **Name**: Student name (text).
    3. **Marks**: Marks obtained (integer).
- Ensure **data integrity** by using RollNo as a primary key, preventing duplicate entries.

3. Backend Development (Flask Application)

- Initialize the **Flask app** to handle HTTP requests.
- Implement different **routes** for CRUD operations:
    - **Insert**: A form to enter new student data.
    - **Update**: A form to update student details using RollNo.
    - **Delete**: Delete student records by entering the RollNo.
    - **Show**: Display all or specific student records.
- Connect the Flask app to the **MySQL database** for performing SQL queries (like INSERT, UPDATE, DELETE, and SELECT).

4. Frontend Development (HTML Templates)

- Design **simple HTML forms** for each CRUD operation.
- Use an index page to navigate to Insert, Update, Delete, and Show functionalities.
- Display student records using HTML tables for easy readability.
- Implement **form validation** to prevent errors (e.g., check for empty fields or invalid inputs).

5. Testing the Application

- Test **each route** (Insert, Update, Delete, Show) to ensure that the data is correctly managed in the MySQL database.
- Perform **input validation tests** to handle invalid Roll Numbers or missing data.
- Verify that **duplicate entries are not allowed** (due to the primary key constraint).
- Ensure that the system handles **database errors gracefully** (e.g., invalid queries or failed connections).

6. Deployment

- Run the Flask app locally to ensure all components work as expected.
- Access the app through the browser using **localhost (http://127.0.0.1:5000/)**.
- Optionally, deploy the application to a **cloud platform** like Heroku or PythonAnywhere for wider accessibility.

7. Maintenance and Future Enhancements

- Regularly monitor the application for bugs or performance issues.
- Future improvements could include:
    - **User authentication** to restrict access to the system.
    - **Search and filter functionalities** to view student records more efficiently.
    - **Reports generation** to analyze student performance.

---

4. Conclusion

4.1 Summary

The **Student Management System** project is a web-based application developed using the Flask framework and MySQL database, designed to efficiently handle student records. Its primary goal is to offer CRUD (Create, Read, Update, Delete) operations to manage student data, such as roll numbers, names, and marks.

4.2 Key Findings

- **Efficient Data Handling**: The system simplifies managing student records through CRUD operations, ensuring effective data entry, update, deletion, and display.
- **Seamless Database Integration**: By integrating Flask with MySQL, the system ensures reliable storage and retrieval of student data with Roll Number as a unique identifier.
- **User-Friendly Interface**: The system provides an intuitive web interface, allowing users to interact with the database effortlessly.

4.3 Future Scope

- **User Authentication**: Add login and authentication features to restrict access to authorized users, enhancing security for managing student data.
- **Role-Based Access Control**: Implement different user roles (e.g., admin, teacher, student) with varying levels of access to sensitive data and operations.
- **Advanced Search and Filtering**: Enable filtering of student records based on criteria like marks, names, or grades, making data retrieval more efficient.

4.4 Conclusion

In conclusion, the **Student Management System** built using Flask and MySQL successfully streamlines the management of student records by providing easy-to-use CRUD operations. The system ensures data integrity through proper validation, offers real-time updates, and features a user-friendly interface. With its scalability potential, the project can be enhanced further with additional features such as user authentication, reporting, and advanced search functionalities, making it a solid foundation for more comprehensive student data management solutions