

STUDENT MANAGEMENT SYSTEM

A MINI PROJECT REPORT

Submitted by

SHYAM S **220701508**

GURUBARAN T **220701522**

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

THANDALAM

CHENNAI-602105

2023 - 24

BONAFIDE CERTIFICATE

Certified that this project report “**STUDENT MANAGEMENT SYSTEM**” is the bonafide work of “**SHYAM S (220701508), GURUBARAN T (220701522)**” who carried out the project work under my supervision.

Submitted for the Practical Examination held on _____

SIGNATURE

Mrs.K. MAHESMEENA
Assistant Professor,
Computer Science and Engineering
Rajalakshmi Engineering College,
(Autonomous),
Thandalam, Chennai - 602 105

INTERNAL EXAMINER

EXTERNAL EXAMINER

ABSTRACT

This project aims to develop a Student Management System (SMS) using Python and SQL, designed to streamline and automate the administrative tasks associated with managing student information in educational institutions. The system leverages the power of Python for the application logic and SQL for robust data storage and retrieval. The Student Management System provides a comprehensive solution for handling various aspects of student data management. Key functionalities include student registration, updating student profiles, managing course enrollments, tracking academic progress, and generating reports. The system ensures data integrity and security by implementing authentication and authorization mechanisms, allowing only authorized personnel to access and modify sensitive information.

TABLE OF CONTENTS

S.NO	TITLE	PAGE. NO
1.	INTRODUCTION	2
	1.1. OBJECTIVES	2
	1.2. MODULE	2
2.	SURVEY OF TECHNOLOGIES	3
	2.1.SOFTWARE DESCRIPTION	3
	2.2. LANGUAGE	3
	2.2.1. SQL	3
	2.2.2. PYTHON	4
3.	REQUIREMENTS AND ANALYSIS	4
	3.1 REQUIREMENT SPECIFICATION ⁵	5
	3.2 HARDWARE AND SOFTWARE REQUIREMENTS ⁶	5
	3.3 DATA FLOW DIAGRAM ⁶	6
	3.4 DATA DICTIONARY ⁶	7
	3.5 ER-DIAGRAM ⁶	8
	3.6 NORMALIZATION ⁶	8
4.	PROGRAM CODE	11
5.	RESULT AND DISCUSSION	24
6.	TESTING	29
7.	CONCLUSION	30
8.	FUTURE ENHANCEMENTS	30

1. INTRODUCTION

A Student Management System (SMS) is a vital tool for educational institutions to efficiently handle various administrative tasks related to students. By leveraging Python for its robust programming capabilities and SQL for powerful database management, this system can streamline processes such as student enrollment, attendance tracking, grade management, and more. Python's versatility allows for seamless integration with SQL databases, ensuring secure and efficient storage, retrieval, and manipulation of student data. Implementing an SMS using these technologies not only enhances administrative efficiency but also improves the overall management and accessibility of student information.

1.1. OBJECTIVES

The objective of developing a Student Management System using Python and SQL is to create a robust and efficient platform that streamlines the administration and management of student-related information. This system aims to automate routine tasks such as student enrollment, attendance tracking, grade recording, and report generation, thereby reducing manual workload and minimizing errors. By leveraging Python's versatile programming capabilities and SQL's powerful database management features, the system will ensure secure, scalable, and reliable data handling. Ultimately, this project seeks to enhance the overall educational experience by providing administrators, teachers, and students with a user-friendly interface that supports effective communication and data-driven decision-making.

1.2. MODULES

- Database Module
- Student Management Module
- Course Management Module
- Grade Management Module
- Attendance Management Module
- Enroll Management Module

2. SURVEY OF TECHNOLOGIES

2.1. SOFTWARE DESCRIPTION

PyCharm is an Integrated Development Environment (IDE) specifically designed for Python development. PyCharm provides a comprehensive set of tools for coding, debugging, testing, and deploying Python applications. It offers a user-friendly interface and a wide range of features. One of the key features of PyCharm is its powerful code editor, which supports syntax highlighting, code completion, and code analysis, helping developers write clean and error-free code more efficiently. It also comes with built-in support for version control systems like Git, making it easy to manage code repositories directly from the IDE.

2.2. LANGUAGE

2.2.1. SQL

SQL, or Structured Query Language, is a powerful and widely-used programming language designed for managing and manipulating relational databases. It provides a standardized way to interact with databases, allowing users to perform various tasks such as querying data, updating records, and defining database structures. SQL is essential for creating, retrieving, updating, and deleting data from databases. One of the key features of SQL is its ability to perform complex queries on large datasets efficiently. Using SQL, users can write queries to retrieve specific information from databases based on specified criteria, such as selecting all students enrolled in a particular course or calculating the average grade for a group of students. SQL also provides mechanisms for data manipulation, including inserting new records into a database, updating existing records, and deleting unwanted data. This makes it a versatile tool for managing data throughout its lifecycle. Additionally, SQL allows for the creation and management of database structures such as tables, indexes, views, and stored procedures. These structures help organize and optimize data storage and retrieval, ensuring efficient operation of the database system.

2.2.2. PYTHON

Python is a high-level, interpreted programming language known for its simplicity and readability. Guido van Rossum released Python in 1991, and since then, it has gained immense popularity in various fields, including web development, data analysis, artificial intelligence, and scientific computing. One of Python's key strengths lies in its clean and concise syntax, which allows developers to write code that is easy to understand and maintain. Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming, making it versatile for a wide range of applications. It comes with a comprehensive standard library that provides ready-to-use modules and functions for tasks such as file I/O, networking, and data processing, reducing the need for external dependencies. Python's dynamic typing and automatic memory management simplify development, as programmers do not need to declare variable types explicitly, and memory allocation and deallocation are handled by the interpreter. This feature contributes to Python's flexibility and ease of use, especially for beginners.

3. REQUIREMENTS AND ANALYSIS

3.1. REQUIREMENT SPECIFICATION

The Student Management System (SMS) aims to streamline administrative tasks related to managing students, courses, grades, and attendance within an educational institution.:

- **Student Management:**

The system shall allow administrators to add, edit, and delete student records. Student records shall include information such as name, ID, contact details, and other relevant personal information.

- **Course Management:**

Administrators shall have the capability to manage courses by adding, editing, and deleting course information. Course records shall include details such as course code, title, description, and other relevant information.

- **Enrollment Management:**

The system shall facilitate the enrollment process, allowing students to enroll in courses. Administrators shall have the ability to manage student enrollments, including adding or dropping courses for students. Students shall be able to view their current course enrollments.

- **Grades Management:**

The system shall enable teachers to assign grades to students for each course. Teachers shall have the ability to view and update student grades throughout the semester.

- **Attendance Tracking:**

The system shall provide functionality for teachers to mark and track student attendance for each class session. Teachers shall have access to attendance records and be able to generate attendance reports. Students shall be able to view their attendance records.

3.2. HARDWARE AND SOFTWARE REQUIREMENTS

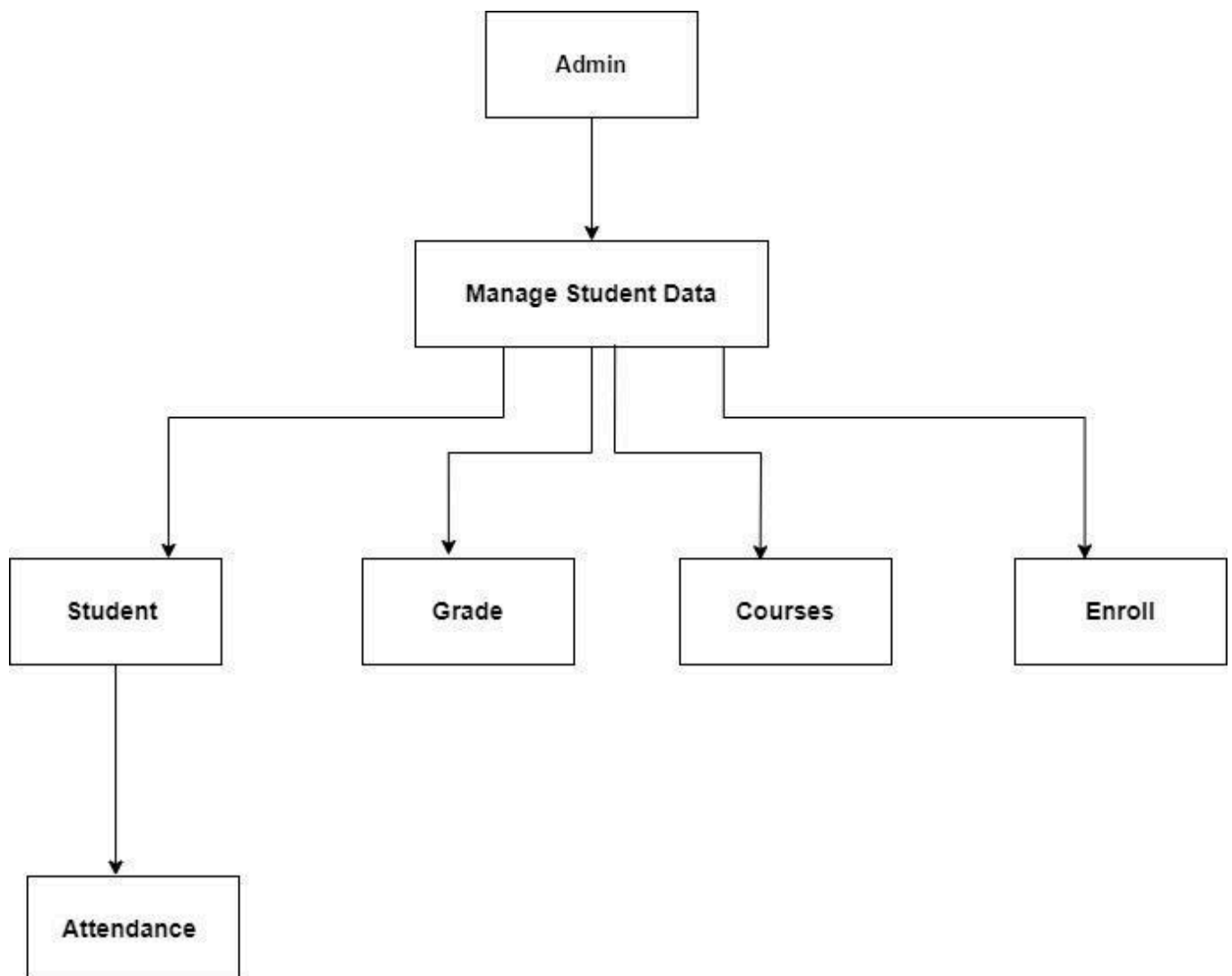
Software Requirements

- Operating System Windows 11
- Front End: Python
- Back End: MySQL

Hardware Requirements

- Desktop PC or a Laptop
- Operating System – Windows 10 , 64-bit operating system
- Intel® Core™ i3-6006U CPU @ 2.00GHz
- 4.00 GB RAM
- Keyboard and Mouse

3.3. DATA FLOW DIAGRAM



3.4. DATA DICTIONARY

- **student**

Column Name	Data Type	Description
student_id	INT	Primary key, auto-incremented student ID
first_name	VARCHAR(100)	First name of the student
last_name	VARCHAR(100)	Last name of the student
date_of_birth	DATE	Date of birth of the student
gender	ENUM('Male', 'Female', 'Other')	Gender of the student
email	VARCHAR(100)	Email address of the student
phone_number	VARCHAR(15)	Phone number of the student

- **courses**

Column Name	Data Type	Description
course_id	INT	Primary key, auto-incremented course ID
course_name	VARCHAR(100)	Name of the course
description	TEXT	Description of the course
credits	INT	Credits associated with the course

- **enrollments**

Column Name	Data Type	Description
enrollment_id	INT	Primary key, auto-incremented enrollment ID
student_id	INT	Foreign key referencing students(student_id)
course_id	INT	Foreign key referencing courses(course_id)
enrollment_date	DATE	Date of enrollment

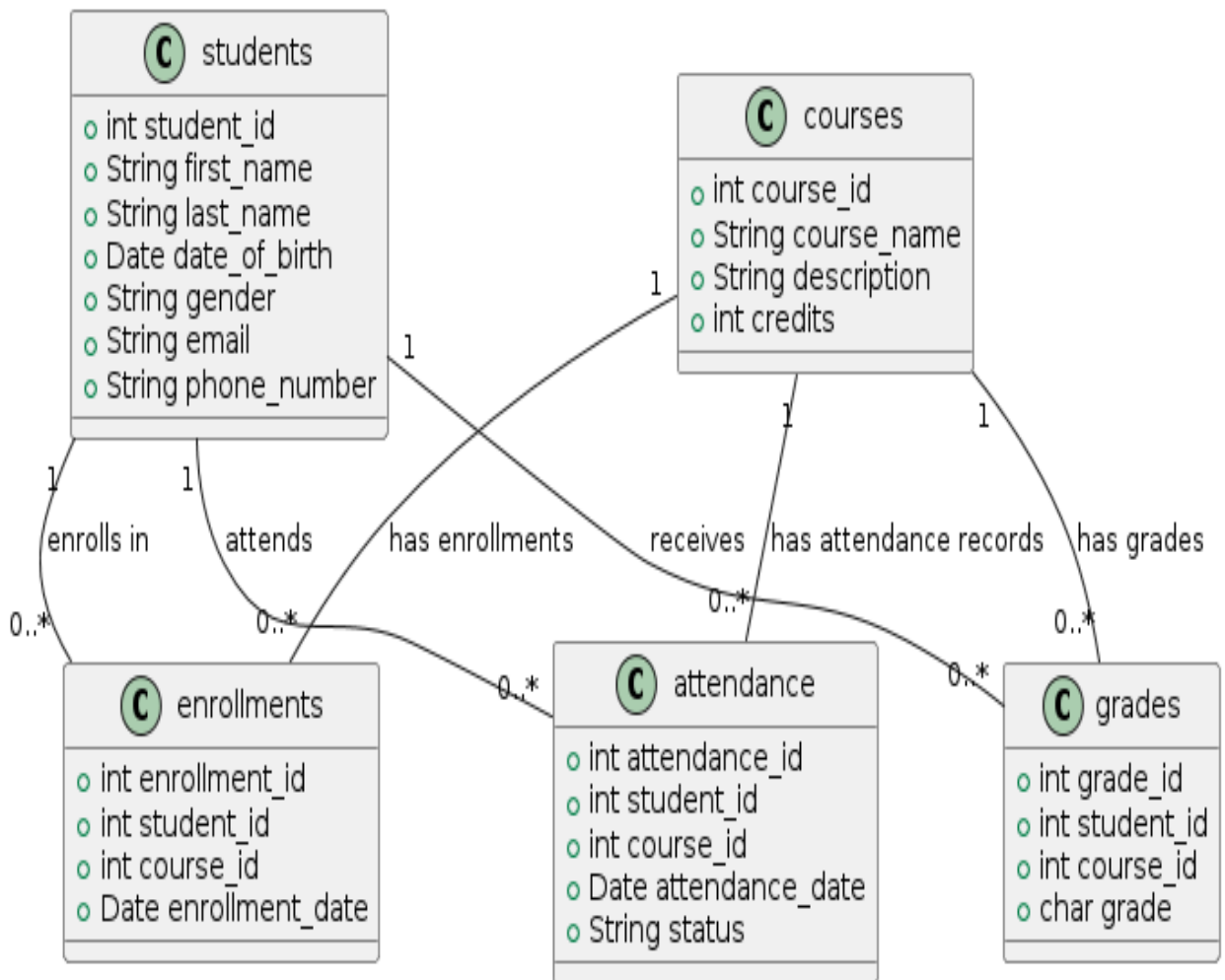
- **grades**

Column Name	Data Type	Description
grade_id	INT	Primary key, auto-incremented grade ID
student_id	INT	Foreign key referencing students(student_id)
course_id	INT	Foreign key referencing courses(course_id)
grade	CHAR(1)	Grade assigned to the student for the course

- **attendance**

Column Name	Data Type	Description
attendance_id	INT	Primary key, auto-incremented attendance ID
student_id	INT	Foreign key referencing students(student_id)
course_id	INT	Foreign key referencing courses(course_id)
attendance_date	DATE	Date of attendance record
status	ENUM('Present', 'Absent')	Attendance status (Present/Absent)

3.5. ER DIAGRAM



3.6. NORMALIZATION

First Normal Form (1NF)

STUDENT TABLE:

Column Name	Data Type	Constraints
student_id	INT	PRIMARY KEY, AUTO_INCREMENT
first_name	VARCHAR(100)	
last_name	VARCHAR(100)	
date_of_birth	DATE	
gender	ENUM('Male', 'Female', 'Other')	
email	VARCHAR(100)	
phone_number	VARCHAR(15)	

COURSES:

Column Name	Data Type	Constraints
course_id	INT	PRIMARY KEY, AUTO_INCREMENT
course_name	VARCHAR(100)	
description	TEXT	
credits	INT	

GRADES:

Column Name	Data Type	Constraints
grade_id	INT	PRIMARY KEY, AUTO_INCREMENT
student_id	INT	FOREIGN KEY REFERENCES students(student_id)
course_id	INT	FOREIGN KEY REFERENCES courses(course_id)
grade	CHAR(1)	

ATTENDANCE:

Column Name	Data Type	Constraints
attendance_id	INT	PRIMARY KEY, AUTO_INCREMENT
student_id	INT	FOREIGN KEY REFERENCES students(student_id)
course_id	INT	FOREIGN KEY REFERENCES courses(course_id)
attendance_date	DATE	
status	ENUM('Present', 'Absent')	

ENROLLMENTS:

Column Name	Data Type	Constraints
enrollment_id	INT	PRIMARY KEY, AUTO_INCREMENT
student_id	INT	FOREIGN KEY REFERENCES students(student_id)
course_id	INT	FOREIGN KEY REFERENCES courses(course_id)
enrollment_date	DATE	

Second Normal Form (2NF)

STUDENT TABLE:

Column	Data Type	Description
student_id	INT AUTO_INCREMENT PRIMARY KEY	Unique ID for each student
first_name	VARCHAR(100)	First name of the student
last_name	VARCHAR(100)	Last name of the student
date_of_birth	DATE	Date of birth of the student
gender	ENUM('Male', 'Female', 'Other')	Gender of the student
email	VARCHAR(100)	Email of the student
phone_number	VARCHAR(15)	Phone number of the student

COURSES TABLE:

Column	Data Type	Description
course_id	INT AUTO_INCREMENT PRIMARY KEY	Unique ID for each course
course_name	VARCHAR(100)	Name of the course
description	TEXT	Description of the course
credits	INT	Number of credits for the course

ENROLLMENT TABLE

Column	Data Type	Description
enrollment_id	INT AUTO_INCREMENT PRIMARY KEY	Unique ID for each enrollment
student_id	INT	Foreign key referencing students(student_id)
course_id	INT	Foreign key referencing courses(course_id)
enrollment_date	DATE	Date of enrollment
FOREIGN KEY (student_id)		REFERENCES students(student_id)
FOREIGN KEY (course_id)		REFERENCES courses(course_id)

GRADE TABLE

Column	Data Type	Description
grade_id	INT AUTO_INCREMENT PRIMARY KEY	Unique ID for each grade
student_id	INT	Foreign key referencing students(student_id)
course_id	INT	Foreign key referencing courses(course_id)
grade	CHAR(1)	Grade received (e.g., A, B, C, D, F)
FOREIGN KEY (student_id)		REFERENCES students(student_id)
FOREIGN KEY (course_id)		REFERENCES courses(course_id)

ATTENDANCE TABLE

Column	Data Type	Description
attendance_id	INT AUTO_INCREMENT PRIMARY KEY	Unique ID for each attendance record
student_id	INT	Foreign key referencing students(student_id)
course_id	INT	Foreign key referencing courses(course_id)
attendance_date	DATE	Date of attendance
status	ENUM('Present', 'Absent')	Attendance status (present or absent)
FOREIGN KEY (student_id)		REFERENCES students(student_id)
FOREIGN KEY (course_id)		REFERENCES courses(course_id)

4. PROGRAM CODE

PYTHON CODE:

```
import mysql.connector

from mysql.connector import Error

def create_connection():

    return mysql.connector.connect(

        host="localhost",

        user="root",

        password="12345678",

        database="student_management"

    )

def insert_student(first_name, last_name, dob, gender, email, phone):

    connection = create_connection()

    cursor = connection.cursor()

    query = "INSERT INTO students (first_name, last_name, date_of_birth, gender, email, phone_number) VALUES (%s, %s, %s, %s, %s, %s)"

    cursor.execute(query, (first_name, last_name, dob, gender, email, phone))

    connection.commit()

    cursor.close()

    connection.close()
```

```

print("Student inserted successfully.")

def update_student(student_id, first_name, last_name, dob, gender, email, phone):

    connection = create_connection()

    cursor = connection.cursor()

    query = "UPDATE students SET first_name=%s, last_name=%s,
date_of_birth=%s, gender=%s, email=%s, phone_number=%s WHERE student_id=%s"

    cursor.execute(query, (first_name, last_name, dob, gender, email, phone,
student_id))

    connection.commit()

    cursor.close()

    connection.close()

    print("Student updated successfully.")

def delete_student(student_id):

    connection = create_connection()

    cursor = connection.cursor()

    query = "DELETE FROM students WHERE student_id=%s"

    cursor.execute(query, (student_id,))

    connection.commit()

    cursor.close()

    connection.close()

    print("Student deleted successfully.")

```

```
def show_students():

    connection = create_connection()

    cursor = connection.cursor()

    query = "SELECT * FROM students"

    cursor.execute(query)

    rows = cursor.fetchall()

    for row in rows:

        print(row)

    cursor.close()

    connection.close()

def show_courses():

    connection = create_connection()

    cursor = connection.cursor()

    query = "SELECT * FROM courses"

    cursor.execute(query)

    rows = cursor.fetchall()

    for row in rows:

        print(row)

    cursor.close()

    connection.close()
```



```

def show_enrollments():

    connection = create_connection()

    cursor = connection.cursor()

    query = "SELECT * FROM enrollments"

    cursor.execute(query)

    rows = cursor.fetchall()

    for row in rows:

        print(row)

    cursor.close()

    connection.close()

def show_grades():

    connection = create_connection()

    cursor = connection.cursor()

    query = "SELECT * FROM grades"

    cursor.execute(query)

    rows = cursor.fetchall()

    for row in rows:

        print(row)

    cursor.close()

    connection.close()

```

```

def show_attendance():

    connection = create_connection()

    cursor = connection.cursor()

    query = "SELECT * FROM attendance"

    cursor.execute(query)

    rows = cursor.fetchall()

    for row in rows:

        print(row)

    cursor.close()

    connection.close()

# Example usage

def insert_course(course_id, course_name, description, credits):

    connection = create_connection()

    cursor = connection.cursor()

    query = "INSERT INTO courses (course_id, course_name,description,credits)
VALUES (%s, %s, %s, %s)"

    cursor.execute(query, (course_id, course_name, description, credits))

    connection.commit()

    cursor.close()

    connection.close()

    print("course inserted successfully.")

```

```

def Enroll_student(enrollment_id, student_id, course_id, enrollment_date, ):

    connection = create_connection()

    cursor = connection.cursor()

    query = "INSERT INTO enrollments (enrollment_id, student_id,course_id,
enrollment_date) VALUES (%s, %s, %s, %s)"

    cursor.execute(query, (enrollment_id, student_id, course_id, enrollment_date))

    connection.commit()

    cursor.close()

    connection.close()

    print("Enrolled successfully.")

def grade_student(grade_id, student_id, course_id, grade):

    connection = create_connection()

    cursor = connection.cursor()

    query = "INSERT INTO grades (grade_id ,student_id ,course_id ,grade)
VALUES (%s, %s, %s, %s)"

    cursor.execute(query, (grade_id, student_id, course_id, grade))

    connection.commit()

    cursor.close()

    connection.close()

    print("Grade Added successfully.")

def attend_student(attendance_id, student_id, course_id, attendance_date, status):

```

```

connection = create_connection()

cursor = connection.cursor()

query = "INSERT INTO attendance (attendance_id ,student_id,course_id
,attendance_date ,status) VALUES (%s, %s, %s, %s,%s)"

cursor.execute(query, (attendance_id, student_id, course_id, attendance_date,
status))

connection.commit()

cursor.close()

connection.close()

print("Attendance Added successfully.")

if __name__ == "__main__":

    while True:

        print("1. Add Data")

        print("2. Show Data")

        print("3. Exit")

        choice = input("Enter choice: ")

        if choice == '1':

            print("1. Insert student")

            print("2. Update student")

            print("3. Delete student")

            print("4. Show students")

```

```
print("5. Insert Course")

print("6. Enroll student")

print("7. Grade")

print("8. Attendance")

print("9. Exit")

student_choice = input("Enter choice: ")

if student_choice == '1':

    first_name = input("Enter first name: ")

    last_name = input("Enter last name: ")

    dob = input("Enter date of birth (YYYY-MM-DD): ")

    gender = input("Enter gender (Male/Female/Other): ")

    email = input("Enter email: ")

    phone = input("Enter phone number: ")

    insert_student(first_name, last_name, dob, gender, email, phone)

elif student_choice == '2':

    student_id = int(input("Enter student ID to update: "))

    first_name = input("Enter first name: ")

    last_name = input("Enter last name: ")

    dob = input("Enter date of birth (YYYY-MM-DD): ")
```

```
gender = input("Enter gender (Male/Female/Other): ")

email = input("Enter email: ")

phone = input("Enter phone number: ")

update_student(student_id, first_name, last_name, dob, gender, email,
phone)

elif student_choice == '3':

    student_id = int(input("Enter student ID to delete: "))

    delete_student(student_id)

elif student_choice == '4':

    show_students()

elif student_choice == '5':

    course_id = int(input("Enter Course ID: "))

    course_name = input("Enter course Name: ")

    description = input("Enter Description: ")

    credits = input("Enter Credits: ")

    insert_course(course_id, course_name, description, credits)

elif student_choice == '6':

    enrollment_id = int(input("Enter Enrollment ID: "))

    student_id = input("Enter Student ID : ")

    course_id = input("Enter Course ID: ")

    enrollment_date = input("Enter Enrollment date: ")
```

```

        Enroll_student(enrollment_id, student_id, course_id, enrollment_date)

elif student_choice == '7':

    grade_id = int(input("Enter Grade ID: "))

    student_id = input("Enter Student ID : ")

    course_id = input("Enter Course ID: ")

    grade = input("Enter Grade: ")

    grade_student(grade_id, student_id, course_id, grade

elif student_choice == '8':

    attendance_id = int(input("Enter Attendance ID: "))

    student_id = input("Enter Student ID : ")

    course_id = input("Enter Course ID: ")

    attendance_date = input("Enter Attendance date: ")

    status = input("Enter Status: ")

    attend_student(attendance_id, student_id, course_id, attendance_date,
status)

elif student_choice == '9':

    break

else:

    print("Invalid choice. Please try again.")

elif choice == '2':

    while True:

```

```
print("1. Show student")

print("2. Show Courses")

print("3. Show Enrollments")

print("4. Show Grades")

print("5. Show Attendance")

print("6. Go back to Main Menu")

data_choice = input("Enter choice: ")


if data_choice == '1':

    show_students()

elif data_choice == '2':

    show_courses()

elif data_choice == '3':

    show_enrollments()

elif data_choice == '4':

    show_grades()

elif data_choice == '5':

    show_attendance()

elif data_choice == '6':

    break
```



```
        else:

            print("Invalid choice. Please try again.")

    elif choice == '3':

        print("Thank you for using Student Management.")

        break

    else:

        print("Invalid choice. Please try again.")
```

MYSQL QUERIES:

```
CREATE DATABASE student_management;
USE student_management;

CREATE TABLE students (

    student_id INT AUTO_INCREMENT PRIMARY KEY,

    first_name VARCHAR(100),

    last_name VARCHAR(100),

    date_of_birth DATE,

    gender ENUM('Male', 'Female', 'Other'),

    email VARCHAR(100),

    phone_number VARCHAR(15)

);

CREATE TABLE courses (

    course_id INT AUTO_INCREMENT PRIMARY KEY,
```

```

course_name VARCHAR(100),

description TEXT,

credits INT

);

CREATE TABLE enrollments (

    enrollment_id INT AUTO_INCREMENT PRIMARY KEY,

    student_id INT,

    course_id INT,

    enrollment_date DATE,

    FOREIGN KEY (student_id) REFERENCES students(student_id),

    FOREIGN KEY (course_id) REFERENCES courses(course_id)

);

CREATE TABLE grades (

    grade_id INT AUTO_INCREMENT PRIMARY KEY,

    student_id INT,

    course_id INT,

    grade CHAR(1),

    FOREIGN KEY (student_id) REFERENCES students(student_id),

    FOREIGN KEY (course_id) REFERENCES courses(course_id)

);

```

```

CREATE TABLE attendance (

attendance_id INT AUTO_INCREMENT PRIMARY KEY,

student_id INT,

course_id INT,

attendance_date DATE,

status ENUM('Present', 'Absent'),

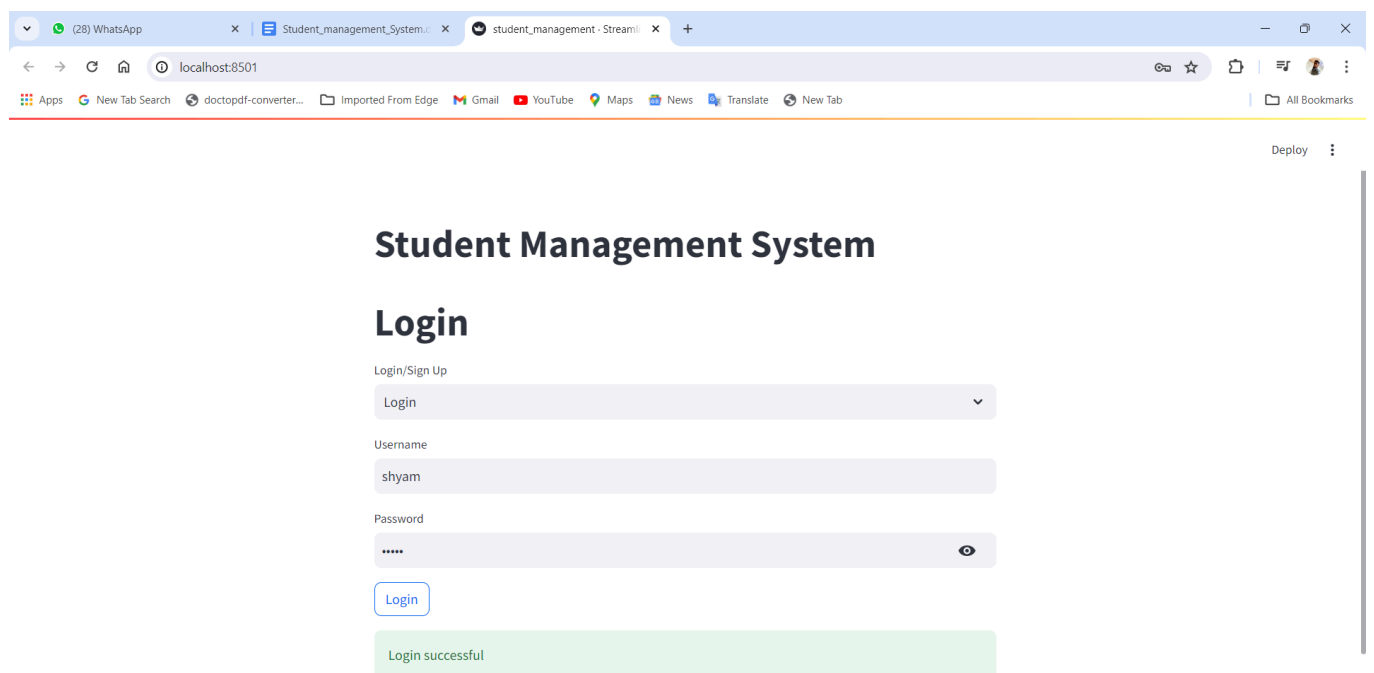
FOREIGN KEY (student_id) REFERENCES students(student_id),

FOREIGN KEY (course_id) REFERENCES courses(course_id)

);

```

5. RESULT AND DISCUSSION



Student Management System

Login

Login/Sign Up

Login

Username

shyam

Password

.....

Login

Login successful

Menu

Add Data

Logout

Student Management System

Add Data

Choose Data to Add

Student

First Name

shyam

Last Name

S

Date of Birth

2024/06/02

Gender

Male

Email

220701522@rajalakshmi.edu.in

Phone Number

9896865447

Insert Student

Menu

Show Data

Logout

Student Management System

Show Data

Choose Data to Show

Students

Search

	student_id	first_name	last_name	date_of_birth	gender	email	phone_n
0	8	Shyam	S	2005-08-10	Male	220701508@rajalakshmi.edu.in	90252385

Menu

Update Data

Logout

Student Management System

Update Data

Select Student

Shyam S (8)

First Name

Last Name

Date of Birth

2024/06/02

Gender

Male

Email

Phone Number

Update Student

Menu

Delete Data

Logout

Student Management System

Delete Data

Select Student

Shyam S (8)

Delete Student

Menu

All Data

Logout

Student Management System

All Data

Search by student_id, name, or date of birth

	student_id	first_name	last_name	date_of_birth	gender	email	phone_n
0	8	Shyam	S	2005-08-10	Male	220701508@rajalakshmi.edu.in	90252385

6. TESTING

6.1. Unit Testing

Unit testing is a testing technique in which modules are tested individually. Small individual units of source code are tested to determine whether it is fit to use or not. Different modules of games are put to test while the modules are being developed. Here modules refer to individual levels, players, scenes.

6.2. Integration Testing

Integration testing is the technique in which individual components or modules are grouped together and tested. It occurs after testing. The inputs for the integrated testing are the modules that have already been unit tested.

6.3. System Testing

System testing is conducted on the entire system as a whole to check whether the system meets its requirements or not. software was installed on different systems and any errors or bugs that occurred were fixed.

6.4. Acceptance Testing

User Acceptance is defined as a type of testing performed by the Client to certify the system with respect to the requirements that was agreed upon. This testing happens in the final phase of testing before moving the software application to the Market or Production environment.

7. CONCLUSION

In conclusion, the development of a Student Management System using Python and SQL provides a robust, efficient, and scalable solution for managing academic data. This system streamlines the process of storing, retrieving, and updating student records, making administrative tasks more manageable and less error-prone. By leveraging the versatility of Python for backend logic and the reliability of SQL for database management, the system ensures data integrity, security, and accessibility. The integration of these technologies not only enhances the user experience but also prepares the groundwork for future enhancements, such as incorporating data analytics and expanding to web-based interfaces. Ultimately, this project demonstrates the potential of combining Python and SQL to create practical, real-world applications that address specific needs in educational institutions.

8. FUTURE ENHANCEMENTS:

- **Integration with Online Learning Platforms:** Allowing seamless integration with e-learning platforms like Moodle or Google Classroom to provide a more holistic educational experience.
- **Mobile Application Development:** Creating a mobile app version to provide accessibility on the go, making it easier for students and administrators to access information from anywhere.
- **Advanced Analytics:** Implementing advanced data analytics and machine learning algorithms to provide predictive insights into student performance and identify areas needing intervention.
- **Cloud-Based Solutions:** Migrating the system to a cloud-based infrastructure to enhance accessibility, scalability, and collaboration across different geographical locations.