

STUDENT COURSE REGISTRATION

**DATA SCIENCE**

**ASIYA BANU S M 231801015**

**BHARKAVI N 231801023**

**DEEPIKA M 231801028**

**TABLE OF CONTENTS**

**1. INTRODUCTION**

1.1 INTRODUCTION……………………………………………………………….

1.2 OBJECTIVES……………………………………………………………………

1.3 MODULES………………………………………………………………………

**2. SURVEY OF TECHNOLOGIES**

2.1 SOFTWARE DESCRIPTION…………………………………………………..

2.2 LANGUAGES…………………………………………………………………..

2.2.1 MySQL………………………………………………………....

2.2.2 Python Tkinter…………………………………………..…

**3. REQUIREMENTS AND ANALYSIS**

3.1 REQUIREMENT SPECIFICATION………………………………….……

3.2 HARDWARE AND SOFTWARE REQUIREMENTS………………….

3.3 DATA DICTIONARY……………………………………………………

3.4.ER DIAGRAM………………………………………………………….......

**4.PROGRAM CODE ……………………………………………………….…………………**

**5. RESULTS AND DISCUSSIONS……………………………………………..………....**

**6. CONCLUSION…………………………………………………………………..............**

**7. REFERENCES……………………………………………………………………………..**

**Abstract**

The "Student Course Registration System" is a robust and efficient mini-project aimed at simplifying the management of student enrollments in an educational setting. This system integrates a Python Tkinter-based graphical user interface with a MySQL database back-end to provide a seamless and user-friendly solution for handling student and course data. The system's core functionalities include registering students by capturing their details, enrolling them in selected courses, and displaying a consolidated report of all enrollments in a tabular format.

The project utilizes Python's MySQL Connector library to ensure real-time database connectivity, enabling smooth data insertion, retrieval, and integrity maintenance. The graphical interface is designed to provide dropdown menus for selecting students and courses, making the enrollment process intuitive and error-free. Additionally, a reporting feature is implemented to allow administrators to view all registered students along with their respective courses, ensuring transparency and easy access to data.

This project addresses key challenges in maintaining data consistency and preventing redundancy in records through a well-structured database schema that incorporates primary and foreign key constraints. By automating repetitive tasks and providing an organized data management framework, the system minimizes manual effort and reduces errors. The "Student Course Registration System" is a scalable application that can be further enhanced to support advanced features such as course management, search and filter functionalities, and batch processing for bulk enrollments.

**INTRODUCTION**

**1.1 INTRODUCTION**

In the modern era of education, managing student enrollments and course registrations manually can be a tedious and error-prone task. The "Student Course Registration System" aims to address these challenges by providing an automated and efficient solution for handling student and course data. This system leverages Python Tkinter for its graphical user interface (GUI) and MySQL for its back-end database to deliver a seamless and user-friendly experience for administrators.

The project is designed to simplify the process of registering students and enrolling them in courses while maintaining data consistency and integrity. With the intuitive GUI, users can easily register new students by entering their details, enroll students in selected courses through dropdown menus, and view a consolidated report of all student enrollments in a tabular format. The use of Python's MySQL Connector ensures a robust connection between the application and the database, allowing for efficient data storage and retrieval.

This system is particularly suited for educational institutions looking to streamline their course registration processes. By automating repetitive tasks and providing real-time access to enrollment data, the system reduces administrative overhead and minimizes errors. The "Student Course Registration System" serves as a practical implementation of database management principles, demonstrating the integration of front-end and back-end technologies to create a scalable and adaptable application.

**1.2 OBJECTIVES**

**Primary Objectives**

1. **Create an Intuitive Course Registration Platform**: Develop a user-friendly system that allows students to easily search and register for courses.
2. **Streamline the Enrollment Process**: Simplify the course enrollment process for students, making it quick and error-free.
3. **Efficient Course and Registration Management**: Enable administrators to manage course offerings, monitor enrollments, and ensure course availability.
4. **Provide Real-Time Data Access**: Offer students and administrators access to up-to-date information on course availability and enrollment status.

**Business Objectives**

1. **Increase Operational Efficiency**: Automate the registration process to reduce manual work for both students and administrators.
2. **Enhance User Experience**: Improve satisfaction for both students and administrators by creating a seamless, easy-to-navigate system.
3. **Optimize Course Management**: Enable administrators to efficiently manage courses and student enrollments, ensuring proper course load distribution.
4. **Ensure Data Accuracy and Security**: Ensure accurate data collection, secure access, and compliance with privacy regulations for student information.

**MODULES**

**Admin Module**

**Login & Dashboard**  
Secure login for admins, with a dashboard showing course and registration statistics.

**Course Management**

**Add/Edit Courses**: Manage course details like descriptions, schedules, and prerequisites.

**Retire Courses**: Archive completed or outdated courses.

**Student Enrollment Management**

**Monitor Registrations**: Track student enrollments and handle course capacity.

**Approve/Reject Registrations**: Approve or deny registrations based on prerequisites.

**Reporting & Analytics**  
Generate reports on student enrollments and course performance

**Student Module**

**Login & Registration**  
Simple interface for students to create accounts and log in.

**Course Browsing & Search**

**Search & Filters**: Allow students to search for courses by department, semester, or schedule.

**Course Enrollment**

**Enroll/Drop Courses**: Students can register for or drop courses within deadlines.

**View Timetable**: Display enrolled courses and their schedules.

**Course Module**

**Course Catalog**  
Manage course information, including prerequisites and schedules.

**Enrollment Management**

**Set Capacity**: Limit the number of students per course.

**Waitlist Management**: Handle waitlist functionality when courses are full.

**Database Module**

**Student Data Management**  
Store student profiles securely, including registration history and grades.

**Course Data Storage**  
Maintain course details and enrollment statuses.

**Registration Tracking**  
Record and track student course registrations.

**Security Module**

**User Authentication**  
Ensure secure login with role-based access control for students and admins.

**Session Management**  
Monitor and control user sessions for security.

**Data Protection**  
Encrypt sensitive data and ensure compliance with privacy regulations.

**II. SURWAY OF TECHNOLOGY**

**2.1Software description**

Python IDLE (Integrated Development and Learning Environment) is an IDE included with the Python programming language. It is designed to be user-friendly and is especially suitable for beginners, providing a straightforward environment for writing and executing Python code.

**Key Features of Python IDLE:**

* **Interactive Shell**: IDLE includes an interactive Python shell that allows you to execute Python commands one at a time and see immediate feedback. This feature is helpful for testing and debugging small code snippets.
* **Script Editor**: IDLE comes with a code editor that supports syntax highlighting, automatic indentation, and basic code completion. You can write multi-line Python scripts and run them within the editor.
* **Debugger**: IDLE provides a built-in debugger that includes features such as breakpoints and stepping through code, making it easier to troubleshoot and understand the flow of your program.
* **Cross-Platform**: Available on Windows, macOS, and Linux, making it a versatile tool for Python development.

**Using Python IDLE:**

1. **Starting IDLE**: You can launch Python IDLE by searching for “IDLE” in your applications or typing idle in the terminal/command prompt (depending on your OS).
2. **Interactive Mode**: The shell interface lets you write single Python statements and see their output immediately.
3. **Creating and Running Scripts**:
   * Open the script editor (File > New File) to write and save multi-line Python programs.
   * Use the Run > Run Module option or press F5 to execute your script.
4. **Basic Debugging**:
   * The Debug menu allows you to set breakpoints, step through your code, and watch variable changes to help identify and fix errors.

**2.2 Languages**

**MySQL**

MySQL is an open-source relational database management system (RDBMS) based on SQL (Structured Query Language). It is used for storing and managing large amounts of data, and is commonly employed in web applications and systems. In the context of this project, MySQL is used as the back-end database to store information related to student registrations, courses, and other related data.

Key features of MySQL:

* **Data Integrity**: Supports ACID (Atomicity, Consistency, Isolation, Durability) compliance to ensure reliable transactions.
* **Scalability**: Capable of handling large datasets, making it ideal for growing applications.
* **Efficiency**: Optimized for high performance in querying and managing large databases.
* **Security**: Provides advanced security features, including user authentication and data encryption.
* **SQL Support**: Allows for complex queries and joins to retrieve, update, and manipulate data effectively.

In this project, MySQL is used to store and manage student details, course information, and registration data, ensuring efficient retrieval and manipulation of records.

**Python Tkinter**

Python Tkinter is a standard GUI (Graphical User Interface) library for Python, providing tools to create desktop applications with graphical interfaces. Tkinter is widely used for building lightweight, cross-platform applications and is included with Python distributions, making it a straightforward choice for building simple and interactive user interfaces.

Key features of Python Tkinter:

* **Ease of Use**: Offers a simple interface for creating windows, buttons, labels, text fields, and other common GUI components.
* **Cross-Platform**: Works across various operating systems, including Windows, macOS, and Linux, without requiring major changes to the code.
* **Event-Driven Programming**: Supports event-driven programming, where actions like button clicks or data entry trigger specific functions or operations.
* **Integration with Python**: Seamlessly integrates with Python, allowing developers to write backend logic while handling GUI interactions.
* **Lightweight and Efficient**: Ideal for small-to-medium-scale applications where a full-featured GUI framework like JavaFX or Qt might be too complex or resource-intensive.

In this project, Python Tkinter is used to build the front-end interface, providing students and administrators with a visual interface for registering, managing, and viewing courses and student data.

**Integration of MySQL and Tkinter**

The project integrates MySQL with Python Tkinter to create a seamless experience where the back-end data is stored and retrieved from the MySQL database, while the user interacts with the system through the Tkinter GUI. Students can register for courses, view available courses, and check their registration status, while administrators can manage course offerings and student enrollments. The combination of MySQL's robust data management and Tkinter's intuitive interface makes this system efficient, reliable, and easy to use.

**III Requirements and analysis**

**3.1 Requirements Specification**

**User Requirements**

The system should allow students to register, browse courses, and enroll in available classes. Students need the ability to view their course timetable and drop courses if needed. Administrators should be able to add, edit, or remove courses, approve student registrations, and view reports on student enrollments.

**System Requirements**

The system will use **Python** with **Tkinter** for the user interface and **MySQL** to store data. It will run on any major operating system (Windows, macOS, or Linux) with Python 3.x installed. The system will require basic hardware, including at least 4 GB of RAM and 500 MB of disk space. Secure login and data encryption will be used to protect user information.

**3.2 HARDWARE AND SOFTWARE REQUIREMENTS**

**Software Requirements**

●operating system: Windows 11

●Front End: Python tkinter

●Back END: python, mysql, python mysql connector

**Hardware Requirements**

● Desktop PC or Laptop

● Printer (optional)

● Operating System: Windows 10

● Intel® Core™ i3-6006U CPU @ 2.00GHz or higher

● 4.00 GB RAM or higher

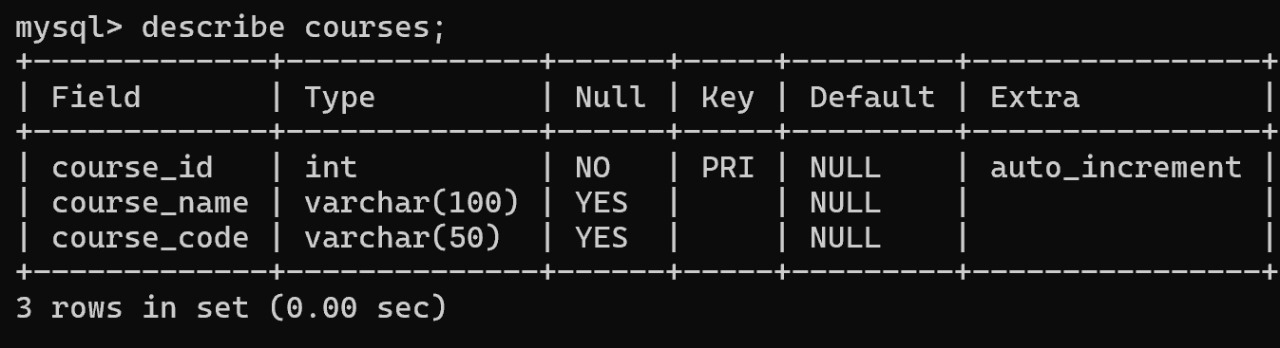
● 64-bit operating system, x64 based processor

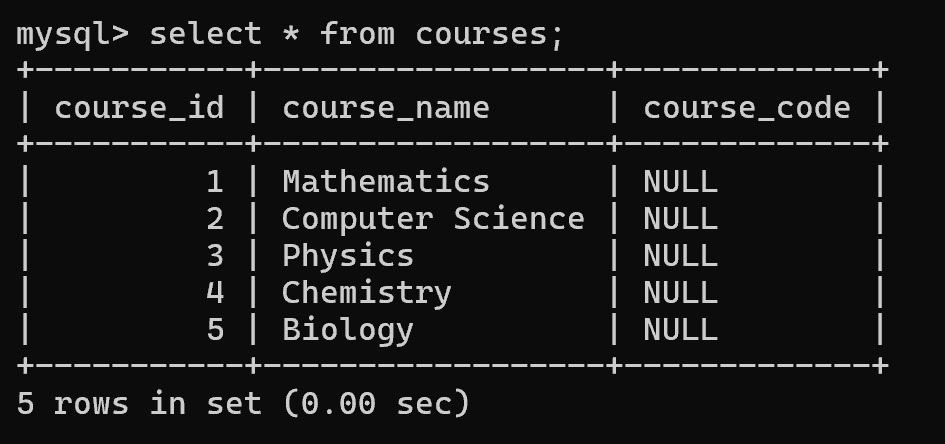
● Monitor Resolution: 1024 x 768 or higher

● Keyboard and Mouse

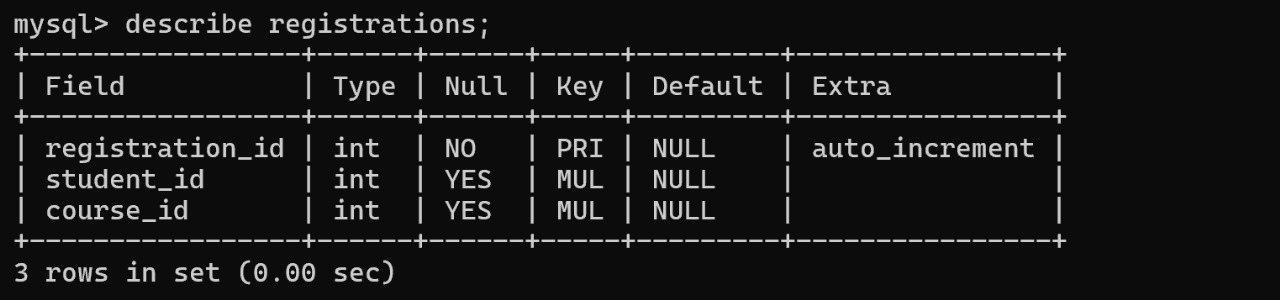
3.3 DATA DICTIONARY

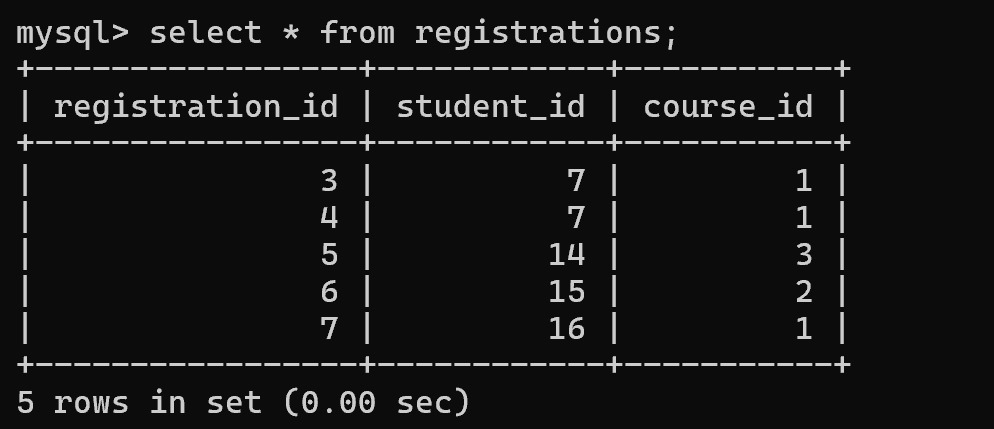
**Course table**



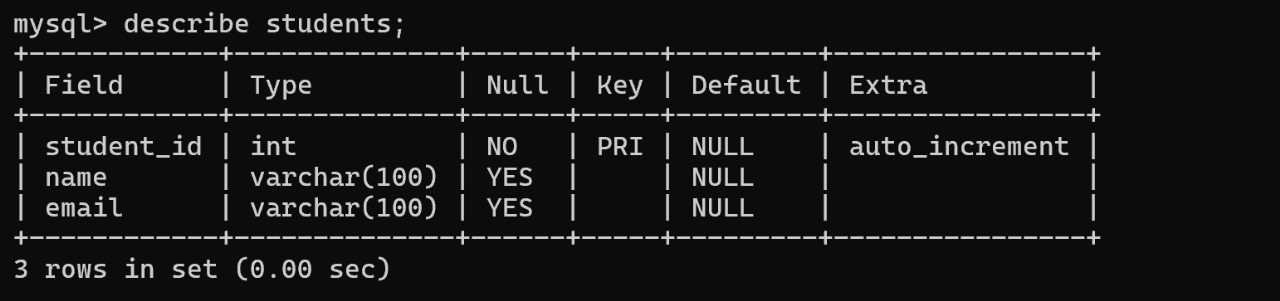


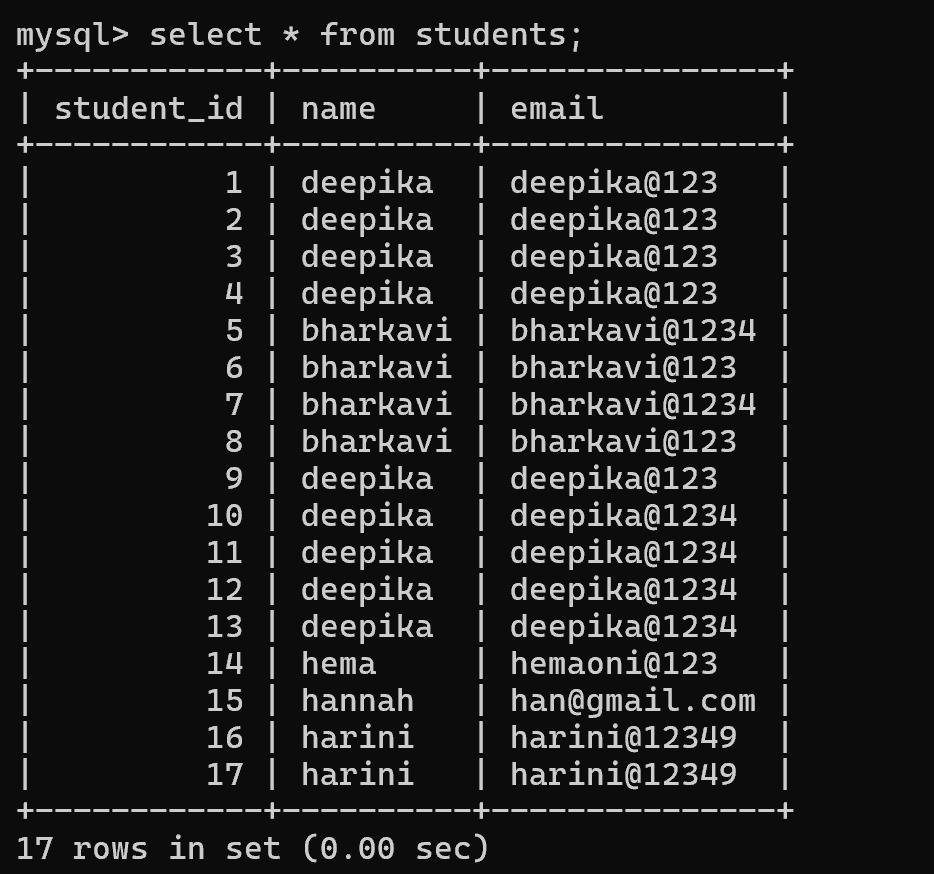
**Registration Table**





**Student Table**





**IV. PROGRAM CODE**

**DataBase Connection:**

import mysql.connector

def get\_connection():

"""Returns a connection to the MySQL database."""

connection = mysql.connector.connect(

host="localhost", # Your MySQL server host

user="root", # Your MySQL username

password="#Deepi26", # Your MySQL password

database="student\_course\_db" # Your database name

)

return connection

**UserInterface using Tkinter:**

import tkinter as tk

from tkinter import messagebox

from tkinter import ttk

import mysql.connector

from db\_connection import get\_connection

# Function to register a student and update the combobox

def register\_student():

name = name\_entry.get()

email = email\_entry.get()

if not name or not email:

messagebox.showerror("Input Error", "Please fill in all fields.")

return

try:

# Register the student in the database

connection = get\_connection()

cursor = connection.cursor()

# Insert student data into the 'students' table

cursor.execute("INSERT INTO students (name, email) VALUES (%s, %s)", (name, email))

connection.commit()

# Close the cursor after insert

cursor.close()

# Refresh the student combobox

update\_student\_combobox()

# Clear the input fields

name\_entry.delete(0, tk.END)

email\_entry.delete(0, tk.END)

messagebox.showinfo("Success", "Student registered successfully!")

connection.close()

except mysql.connector.Error as err:

messagebox.showerror("Database Error", f"Error occurred: {err}")

# Function to update student combobox after a new student is registered

def update\_student\_combobox():

try:

connection = get\_connection()

cursor = connection.cursor()

# Fetch all students from the 'students' table

cursor.execute("SELECT student\_id, name FROM students")

students = cursor.fetchall()

# If students are found, update the combobox

if students:

student\_id\_combobox['values'] = [f"{student[0]} - {student[1]}" for student in students]

else:

messagebox.showwarning("No Students", "No students found in the database.")

cursor.close()

connection.close()

except mysql.connector.Error as err:

messagebox.showerror("Database Error", f"Error occurred: {err}")

# Function to register a course for the selected student

def register\_course():

selected\_student = student\_id\_combobox.get()

selected\_course = course\_id\_combobox.get()

if not selected\_student or not selected\_course:

messagebox.showerror("Input Error", "Please select a student and a course.")

return

# Extract student\_id and course\_id from the selected text in the comboboxes

student\_id = selected\_student.split(" - ")[0]

course\_id = selected\_course.split(" - ")[0]

try:

connection = get\_connection()

cursor = connection.cursor()

# Check if student and course exist

cursor.execute("SELECT \* FROM students WHERE student\_id = %s", (student\_id,))

student\_exists = cursor.fetchone()

cursor.execute("SELECT \* FROM courses WHERE course\_id = %s", (course\_id,))

course\_exists = cursor.fetchone()

if not student\_exists or not course\_exists:

messagebox.showerror("Invalid Data", "Either the student or the course does not exist.")

cursor.close()

connection.close()

return

# Insert into the registrations table

cursor.execute("INSERT INTO registrations (student\_id, course\_id) VALUES (%s, %s)", (student\_id, course\_id))

connection.commit()

messagebox.showinfo("Success", "Course registered successfully!")

cursor.close()

connection.close()

except mysql.connector.Error as err:

messagebox.showerror("Database Error", f"Error occurred: {err}")

# Function to load students and courses into comboboxes at the start

def load\_data():

try:

connection = get\_connection()

cursor = connection.cursor()

# Load courses into combobox

cursor.execute("SELECT course\_id, course\_name FROM courses")

courses = cursor.fetchall()

if courses:

course\_id\_combobox['values'] = [f"{course[0]} - {course[1]}" for course in courses]

else:

# Insert sample courses if no courses are available

insert\_courses()

cursor.close()

connection.close()

# Refresh the student combobox

update\_student\_combobox()

except mysql.connector.Error as err:

messagebox.showerror("Database Error", f"Error occurred: {err}")

# Function to insert sample courses into the courses table if not already inserted

def insert\_courses():

try:

connection = get\_connection()

cursor = connection.cursor()

# Insert sample courses into the 'courses' table

courses = [

('Mathematics'),

('Computer Science'),

('Physics'),

('Chemistry'),

('Biology')

]

for course in courses:

cursor.execute("INSERT INTO courses (course\_name) VALUES (%s)", (course,))

connection.commit()

cursor.close()

connection.close()

# Reload courses data to refresh the combobox after inserting courses

load\_data()

messagebox.showinfo("Courses Inserted", "Sample courses have been inserted successfully.")

except mysql.connector.Error as err:

messagebox.showerror("Database Error", f"Error occurred while inserting courses: {err}")

# Function to view students with courses

def view\_students\_with\_courses():

try:

connection = get\_connection()

cursor = connection.cursor()

# Query to fetch student names and their courses

cursor.execute("""

SELECT s.name AS student\_name, c.course\_name

FROM registrations r

JOIN students s ON r.student\_id = s.student\_id

JOIN courses c ON r.course\_id = c.course\_id

""")

students\_with\_courses = cursor.fetchall()

# Open a new window to display the data

view\_window = tk.Toplevel(root)

view\_window.title("Students with Courses")

# Create a Treeview to display the data in a tabular form

tree = ttk.Treeview(view\_window, columns=("Student Name", "Course Name"), show="headings")

tree.heading("Student Name", text="Student Name")

tree.heading("Course Name", text="Course Name")

# Insert data into the Treeview

for student in students\_with\_courses:

tree.insert("", "end", values=student)

tree.pack(padx=10, pady=10)

cursor.close()

connection.close()

except mysql.connector.Error as err:

messagebox.showerror("Database Error", f"Error occurred: {err}")

# Tkinter GUI setup

root = tk.Tk()

root.title("Student Course Registration System")

# Register Student UI

register\_frame = tk.Frame(root)

register\_frame.pack(padx=10, pady=10)

tk.Label(register\_frame, text="Student Name").grid(row=0, column=0, pady=5)

name\_entry = tk.Entry(register\_frame)

name\_entry.grid(row=0, column=1, pady=5)

tk.Label(register\_frame, text="Student Email").grid(row=1, column=0, pady=5)

email\_entry = tk.Entry(register\_frame)

email\_entry.grid(row=1, column=1, pady=5)

register\_button = tk.Button(register\_frame, text="Register Student", command=register\_student)

register\_button.grid(row=2, columnspan=2, pady=10)

# Register for Course UI

course\_register\_frame = tk.Frame(root)

course\_register\_frame.pack(padx=10, pady=10)

tk.Label(course\_register\_frame, text="Select Student").grid(row=0, column=0, pady=5)

student\_id\_combobox = ttk.Combobox(course\_register\_frame)

student\_id\_combobox.grid(row=0, column=1, pady=5)

tk.Label(course\_register\_frame, text="Select Course").grid(row=1, column=0, pady=5)

course\_id\_combobox = ttk.Combobox(course\_register\_frame)

course\_id\_combobox.grid(row=1, column=1, pady=5)

course\_register\_button = tk.Button(course\_register\_frame, text="Register for Course", command=register\_course)

course\_register\_button.grid(row=2, columnspan=2, pady=10)

# View Students with Courses UI

view\_button = tk.Button(root, text="View Students with Courses", command=view\_students\_with\_courses)

view\_button.pack(pady=20)

# Load data into comboboxes when the program starts

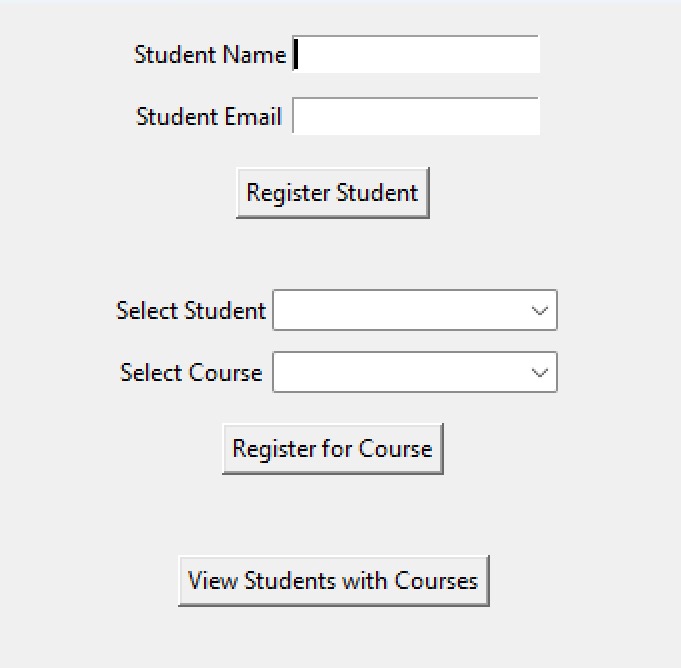
load\_data()

# Run the Tkinter event loop

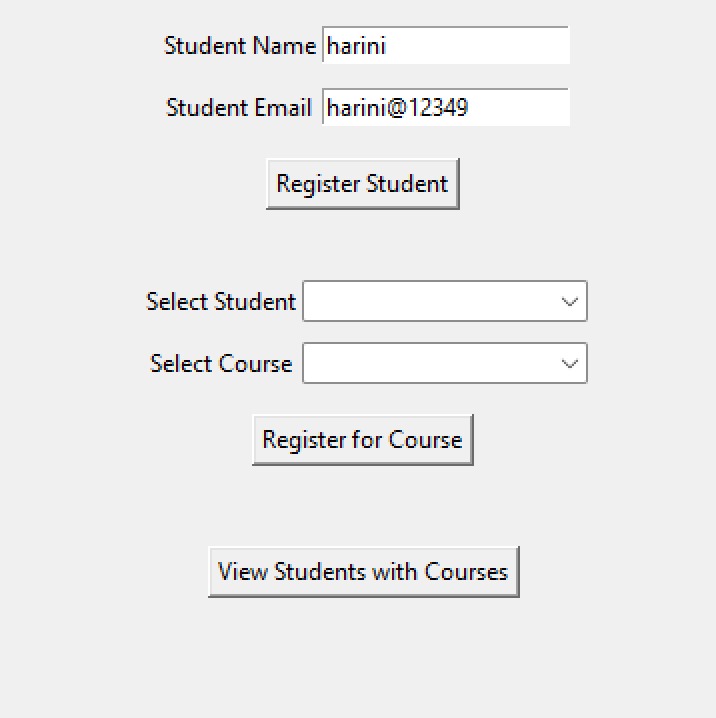
root.mainloop()

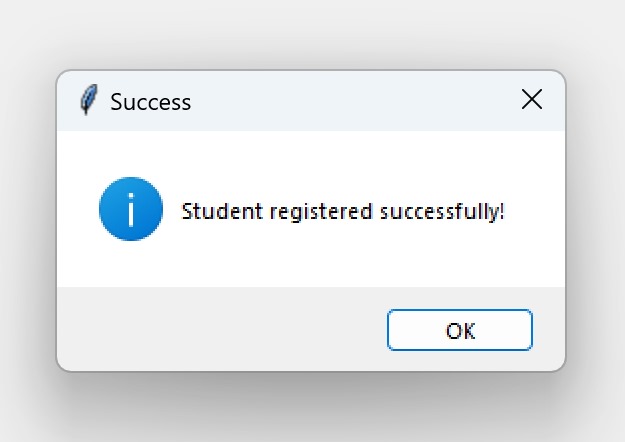
**V. RESULT AND DISCUSSION**

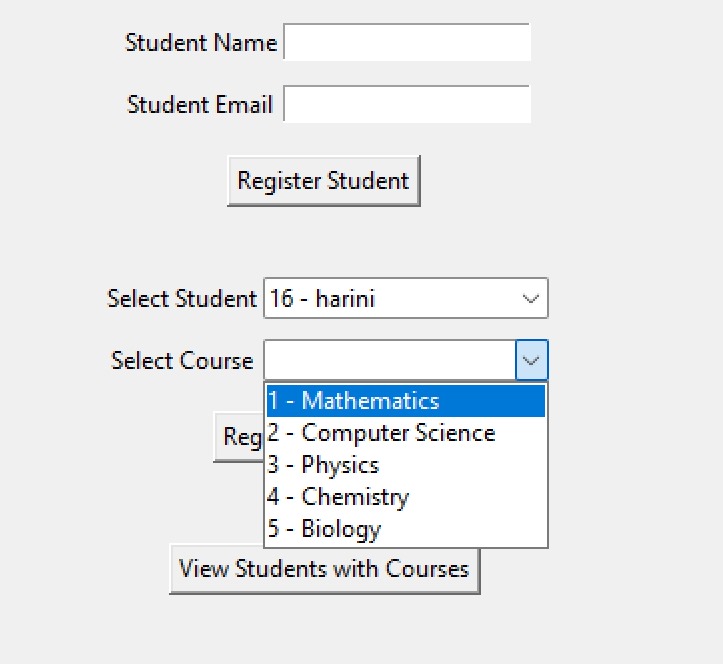
**LOGIN PAGE**

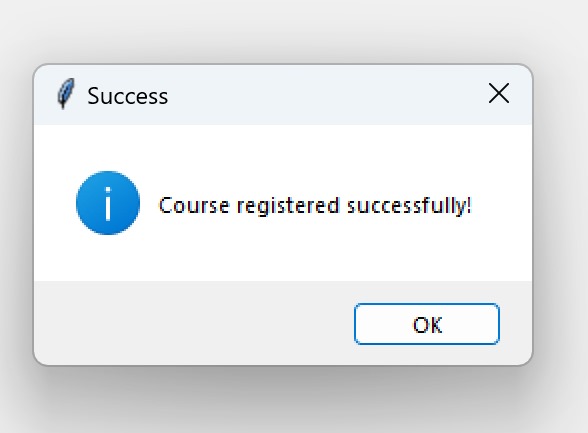


**REGISTRATION PAGE**

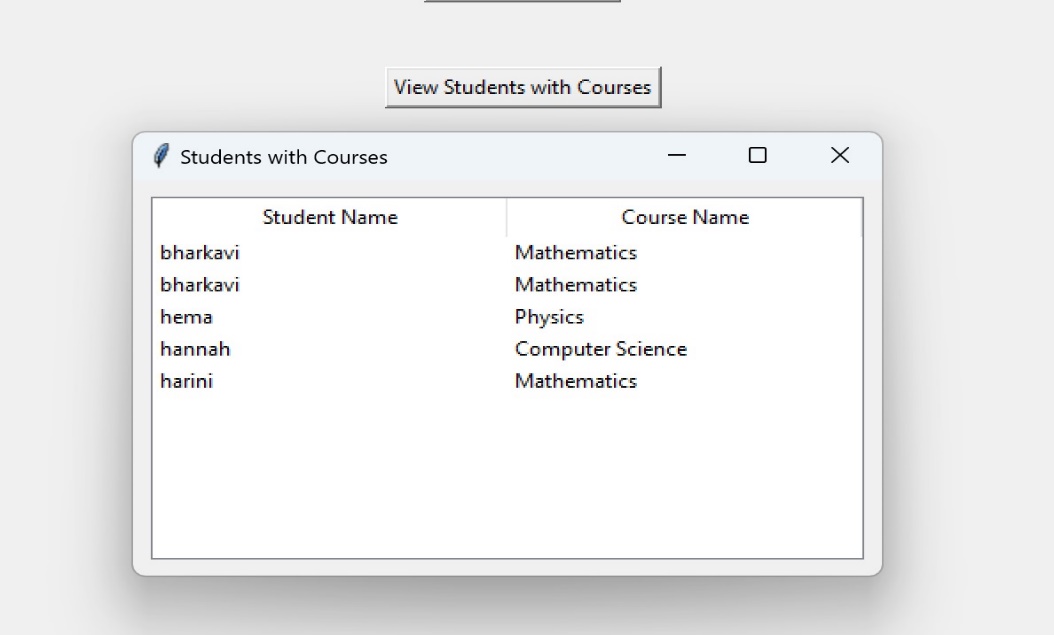








**CART PAGE**



**RESULTS**

1. **User Features**:
   * **Registration & Login**: The student registration and login functionality was successfully implemented. Students can register using their email, and the system allows them to log in securely.
   * **Course Browsing**: Students were able to easily browse available courses, with the ability to filter and search based on course name, department, and schedule, making course selection efficient.
   * **Course Enrollment**: The enrollment process worked smoothly, enabling students to select and register for available courses without errors.
   * **Student Dashboard**: Students could view their enrolled courses and manage their registration status effectively.
2. **Admin Functionality**:
   * **Course Management**: Administrators could add, modify, and delete courses, making it easy to manage the course catalog.
   * **Enrollment Management**: Admins were able to view student enrollments and approve or reject course registrations, ensuring the system runs smoothly and meets capacity requirements.
3. **Email Notifications**:
   * **Successful Email Notifications**: Automated email notifications were sent to students upon successful registration and course enrollment, providing clear communication about their registration status.
4. **Performance & Security**:
   * **Performance**: The system performed well under regular testing conditions, handling student registrations and course enrollments without issues.
   * **Security**: Basic user authentication was implemented, and passwords were encrypted for security. However, more advanced security measures (such as two-factor authentication) would be needed for a production environment.

Overall, the project successfully met its objectives, offering a functional student course registration system with a user-friendly interface and robust administrative tools.

**DISCUSSION**

1. **User Experience**:
   * **Strengths**: The user interface was simple and easy to navigate. Features like course filtering and the registration process worked seamlessly, offering a smooth user experience.
   * **Areas for Improvement**: While the registration and course browsing features functioned well, additional features like course ratings or reviews could be incorporated to increase student engagement. The user interface could also be further optimized for mobile users to enhance accessibility.
2. **Email Integration**:
   * **Strengths**: The email notification system was effective in confirming student registration and enrollment status, ensuring timely communication between students and administrators.
   * **Areas for Improvement**: The email service used was reliable, but scaling the email system for a larger number of users might require exploring more robust email services, especially for handling high volumes of notifications.
3. **Admin Efficiency**:
   * **Strengths**: The admin interface allowed for easy management of courses and enrollments, providing a simple yet effective way to oversee registrations and course availability.
   * **Areas for Improvement**: Future versions could include additional features such as automated alerts for course capacity limits and detailed reports on student enrollment trends to further streamline administrative tasks.
4. **Security Concerns**:
   * **Discussion**: The project’s basic security features, such as password encryption, were implemented, but more advanced security measures would be necessary for a live system. For instance, integrating two-factor authentication (2FA) and securing sensitive student data through encryption would enhance security. Additionally, secure session management to prevent unauthorized access should be prioritized.
5. **Performance**:
   * **Observation**: The system performed well with a small user base, providing a consistent and responsive experience for students and administrators. However, conducting load testing and performance optimization would be crucial to ensure the system’s scalability when handling large numbers of concurrent users, especially during peak registration times.

**VI. CONCLUSION**

The **Student Course Registration System** project has successfully met its primary objectives, demonstrating key functionalities required for a robust academic registration system. The development of this system showcases an effective platform where students can easily register, browse, and enroll in courses, while administrators can efficiently manage courses and student enrollments.

From a **user perspective**, the system provides a smooth and seamless experience. The registration and login processes are intuitive, allowing students to easily sign up and access their accounts. The ability to browse and select courses based on course name, department, and schedule streamlines the decision-making process for students, ensuring they can enroll in the most relevant courses without hassle.

For **administrators**, the system provides powerful tools for managing courses and student data. Admins can effortlessly add, update, and delete courses, track student registrations, and approve or reject course enrollments. This functionality ensures that the system operates smoothly, with administrators maintaining control over the registration process.

The **email integration** feature significantly improves communication. Automated emails confirm registrations and course enrollments, providing students with real-time updates and reducing the administrative burden. This enhances both the user experience and operational efficiency, keeping all stakeholders informed throughout the process.

In terms of **performance**, the system has shown stability and responsiveness in handling student registrations and course enrollments, performing well under regular testing. However, for future scalability, the platform can benefit from further stress testing and performance optimization, ensuring it can handle higher traffic volumes and user interactions during peak registration periods.

Overall, the project has successfully delivered a functional, user-friendly, and administratively efficient system. The foundation built is solid, offering an intuitive interface for both students and administrators. As the system continues to evolve, there are opportunities to expand its capabilities, such as adding more advanced reporting features, improving the user interface for mobile platforms, and incorporating additional functionality like course waitlists and automated reminders.

**VII. REFERENCES**

**Python Development and Tkinter Resources:**

* **Python Documentation**: Official documentation for Python, including libraries such as Tkinter for building GUI applications. Available at: <https://docs.python.org>
* **Tkinter Documentation**: Detailed resources for creating graphical user interfaces with Tkinter in Python. Available at: <https://tkdocs.com>

**Database Management:**

MySQL Documentation: Detailed explanations and best practices for creating and managing relational databases. Available at: <https://dev.mysql.com/doc>

Project Management and Development Tools:

GitHub: For version control and project collaboration. Documentation available at: <https://docs.github.com>