

**DATA SCIENCE**

**Ilakkiya R 231801060**

**Haritha M 231801051**

**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**

A **Salary Management System** is a specialized software solution designed to efficiently manage employee compensation within an organization. It automates the processes of calculating salaries, managing deductions, applying benefits, and ensuring compliance with tax regulations. By replacing manual methods, it reduces the risk of errors, saves time, and enhances overall operational efficiency.

The system centralizes employee data, including attendance, overtime, bonuses, and leaves, to compute accurate salary details. It also facilitates the generation of payslips, tax filings, and financial reports, streamlining the payroll workflow. Additionally, it integrates with banking systems for direct salary disbursement, ensuring timely and secure payments.

A Salary Management System provides a user-friendly interface for HR and finance teams while offering employees access to their payment history and details. With features like data encryption and role-based access, it ensures the confidentiality and security of sensitive financial data. Furthermore, cloud-based systems enable scalability and remote accessibility, making them ideal for organizations of all sizes.

**INTRODUCTION**

* 1. **INTRODUCTION**

A **Salary Management System** is a comprehensive solution designed to streamline and automate the management of employee salaries in an organization. It addresses the complexities of salary computation by integrating various factors, such as employee attendance, overtime, deductions, bonuses, and tax regulations. This system ensures accurate and timely processing of salaries, reducing the administrative burden on HR and finance departments while enhancing overall efficiency.

In traditional setups, salary management involves time-consuming manual processes prone to errors and inconsistencies. A Salary Management System eliminates these challenges by automating payroll workflows, ensuring precise calculations, and maintaining compliance with labor laws and tax requirements. It also provides tools for generating payslips, maintaining financial records, and offering employees access to their salary details through a secure and user-friendly platform.

With features like direct salary disbursement, real-time data updates, and customizable reports, the system supports decision-making and improves financial transparency. Whether deployed in small businesses or large enterprises, it is a scalable and adaptable tool that meets the unique needs of various industries. By integrating with other business systems, such as attendance trackers and accounting software, it creates a unified ecosystem for efficient payroll management.

In summary, a Salary Management System is a vital resource for organizations looking to optimize payroll operations, enhance accuracy, and improve employee satisfaction.

**1.2 OBJECTIVES**

 **Automate Payroll Processes:**  
Streamline the calculation of salaries, deductions, taxes, and benefits to reduce manual effort and errors.

 **Ensure Accuracy:**  
Provide precise and error-free computations by incorporating employee attendance, overtime, bonuses, and leave data.

 **Enhance Efficiency:**  
Minimize the time required for salary processing, report generation, and payment disbursement.

 **Improve Compliance:**  
Ensure adherence to tax laws, labor regulations, and organizational policies by keeping the system updated with regulatory changes.

 **Secure Financial Data:**  
Safeguard sensitive payroll and employee information using robust encryption and access control mechanisms.

 **Generate Reports:**  
Enable easy creation of detailed reports for audits, tax filings, and internal analysis to support decision-making.

 **Facilitate Transparency:**  
Provide employees with easy access to their salary details, payslips, and deductions through a secure portal.

 **Enable Scalability:**  
Adapt to the growing needs of the organization, accommodating an increasing workforce and additional payroll requirements.

 **Integrate with Financial Systems:**  
Seamlessly connect with accounting and banking systems to automate salary transfers and maintain accurate financial records.

 **Reduce Administrative Burden:**  
Free up HR and finance teams to focus on strategic tasks by automating routine payroll activities.

**MODULES**

**Employee Information Management**

* Stores and manages employee data such as personal details, job roles, department, and salary structure.
* Tracks employment history, joining dates, and promotions.

**Attendance and Leave Management**

* Integrates with attendance systems (e.g., biometric or timesheets) to record working hours.
* Tracks leaves, holidays, overtime, and late arrivals to calculate salary adjustments.

**Salary Calculation**

* Automates gross salary calculation by factoring in:
  + Basic pay
  + Allowances (HRA, DA, etc.)
  + Bonuses and incentives
  + Deductions (PF, taxes, loans, etc.)
* Handles prorated salaries for partial months.

**Taxation and Compliance**

* Computes income tax, professional tax, and other statutory deductions.
* Generates Form 16, TDS certificates, and other tax-related documents.
* Ensures compliance with local labor and tax laws.

**II. SURWAY OF TECHNOLOGY**

**2.1Software description**

The **Salary Management System** is a comprehensive software application designed to automate and simplify the management of employee compensation within an organization. It efficiently handles various payroll-related tasks, including salary computation, tax deductions, benefits processing, and compliance with labor laws.

### **Key Features:**

1. **Employee Data Management:** Maintain comprehensive records of employee profiles, salaries, bonuses, and deductions.
2. **Salary Calculation:** Automate gross and net salary calculations, including overtime, bonuses, and deductions.
3. **Tax and Compliance Management:** Stay updated with the latest tax regulations and generate accurate tax reports and filings.
4. **Payslip Generation:** Generate and distribute detailed payslips to employees.
5. **Integration:** Seamlessly integrate with attendance systems and banking platforms.
6. **Reporting and Analytics:** Generate detailed financial reports and payroll analytics to support decision-making.
7. **Security:** Protect sensitive payroll data with encryption and role-based access control.
8. **Scalability:** Adapt to organizations of various sizes and support cloud-based access.

### **Benefits:**

* Reduces manual errors and ensures accurate salary processing.
* Saves time by automating repetitive payroll tasks.
* Enhances employee satisfaction through timely and transparent salary disbursement.
* Improves compliance with legal and tax obligations.
* Offers scalability for growing businesses.

**2.2 Languages**

**MySQL**

**MySQL** is employed to store and manage essential data such as employee details, salary structures, attendance records, and payroll history. Its robust relational database structure allows for efficient data handling and ensures the system remains scalable and reliable.

**Key Features of MySQL in This System:**

1. **Data Integrity**: Supports ACID properties, ensuring accurate and reliable payroll transactions.
2. **Scalability**: Handles large amounts of employee and payroll data as the organization grows.
3. **Efficiency**: Optimized query performance for retrieving and managing salary records.
4. **Security**: Advanced authentication and encryption features protect sensitive financial data.
5. **Flexibility**: SQL queries allow for complex operations such as calculating salaries, generating reports, and tracking attendance.

**Python Tkinter**

**Python Tkinter** serves as the graphical interface, enabling users to interact with the system through windows, buttons, and input fields. Administrators can manage employee data and generate payroll, while employees can view their salary details and payslips.

**Key Features of Python Tkinter in This System:**

**Ease of Use**: Simplifies the creation of forms, data entry screens, and reports.

**Cross-Platform**: Compatible with Windows, macOS, and Linux for wide accessibility.

**Event-Driven Programming**: User actions, such as button clicks or data entry, trigger salary calculations and database updates.

**Integration with Python**: Enables the use of Python libraries for database connectivity and data processing.

**Lightweight**: Ideal for a desktop-based salary management system without the need for heavy frameworks.

**Integration of MySQL and Tkinter**

The integration combines MySQL’s robust data management capabilities with Tkinter’s intuitive user interface to build an efficient salary management system. Python libraries like mysql-connector or pymysql are used to connect the Tkinter front-end with the MySQL database.

**III Requirements and analysis**

**3.1 Requirements Specification**

**User Requirements**

 **Employees**:

* Employees should be able to view their salary details, including components like basic salary, bonuses, and deductions.
* Employees should be able to access their payslips for different months and years.
* Employees can view their attendance status and any leave records affecting their pay.

 **Administrators**:

* Administrators should have the ability to add, edit, and remove employee data.
* Administrators should be able to configure and modify salary components (e.g., basic salary, bonuses, deductions).
* Administrators can generate and view payroll reports for multiple employees at once.
* Administrators should have the ability to process payroll for all employees, ensuring that payments are made on time.

**System Requirements**

* The system will be built using **Python** with **Tkinter** for the graphical user interface and **MySQL** as the back-end database to store employee data, salary information, and payroll records.
* The system should run on any major operating system (Windows, macOS, or Linux) with **Python 3.x** installed.
* **MySQL** should be configured on the system for data storage.
* Basic hardware will include a desktop or laptop with at least **4 GB of RAM** and **500 MB of available disk space**.
* The system should ensure **secure login** for administrators and employees, with **data encryption** to protect sensitive payroll information.

**3.2 HARDWARE AND SOFTWARE REQUIREMENTS**

**Software Requirements**

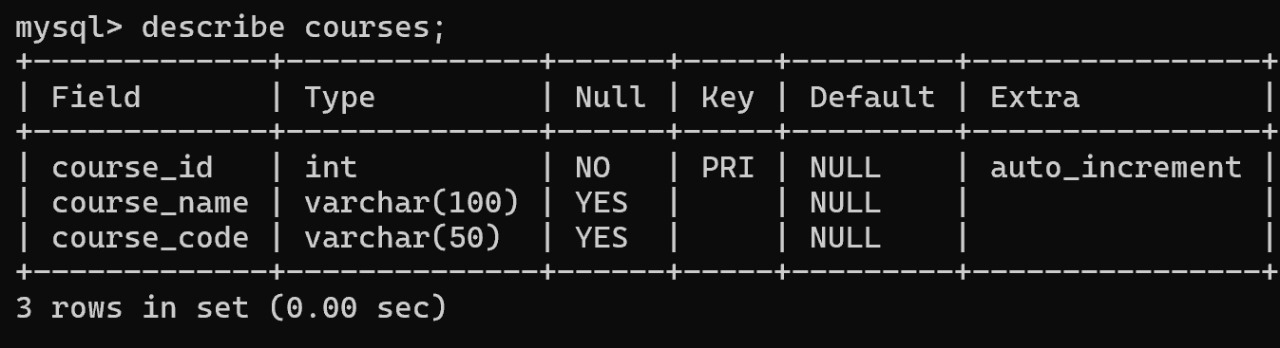
* **Operating System**: Windows 10 or 11, macOS, Linux
* **Front End**: Python (Tkinter)
* **Back End**: Python, MySQL, Python MySQL Connector
* **Database**: MySQL (Version 5.7 or above)
* **Other Tools**: Python 3.x, Python libraries like mysql-connector, tkinter, pandas (for reports)

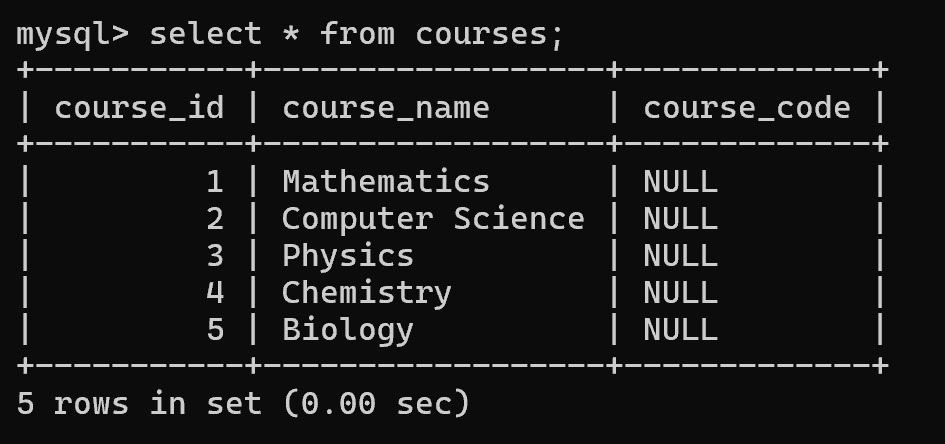
**Hardware Requirements**

* **Hardware**:
  + Desktop PC or Laptop
  + Printer (optional, for printing payslips)
* **System Specifications**:
  + **Operating System**: Windows 10 or higher (64-bit), macOS, or Linux
  + **Processor**: Intel® Core™ i3-6006U CPU @ 2.00GHz or higher
  + **RAM**: 4.00 GB or higher
  + **Disk Space**: 500 MB available disk space
  + **Display**: Monitor resolution of 1024 x 768 or higher
  + **Input Devices**: Keyboard and Mouse

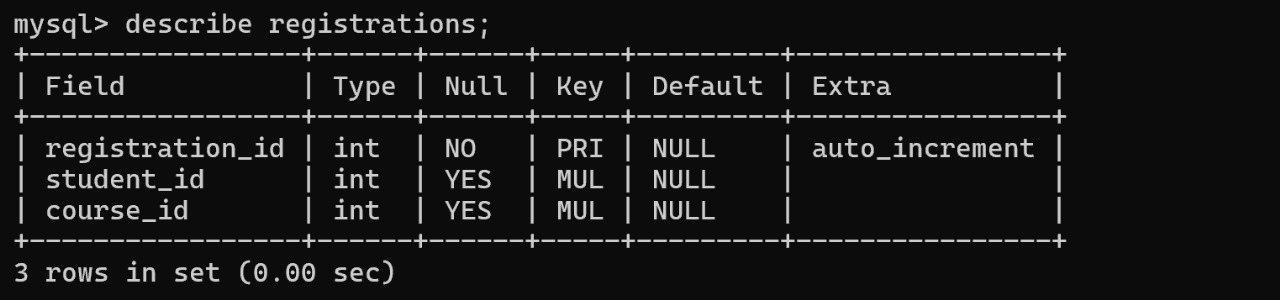
3.3 DATA DICTIONARY

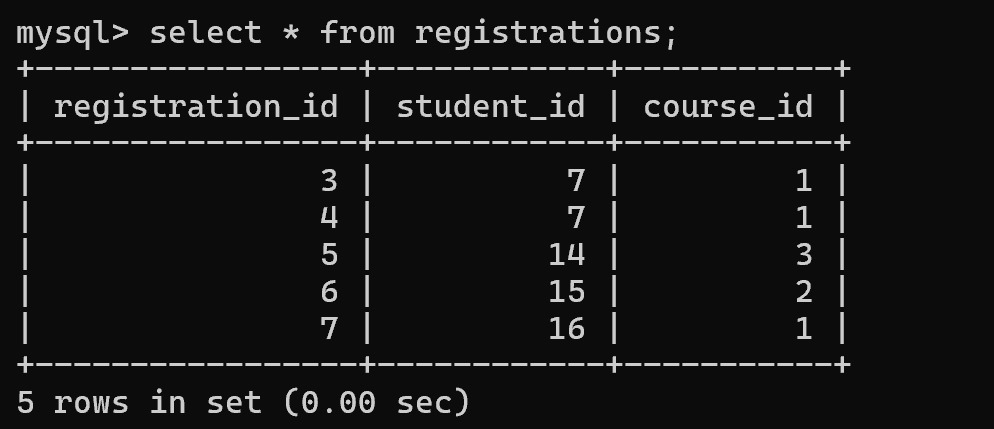
**Course table**



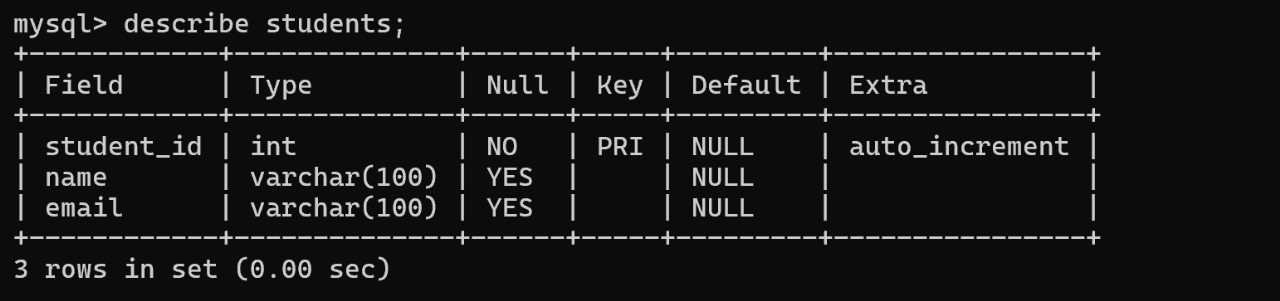


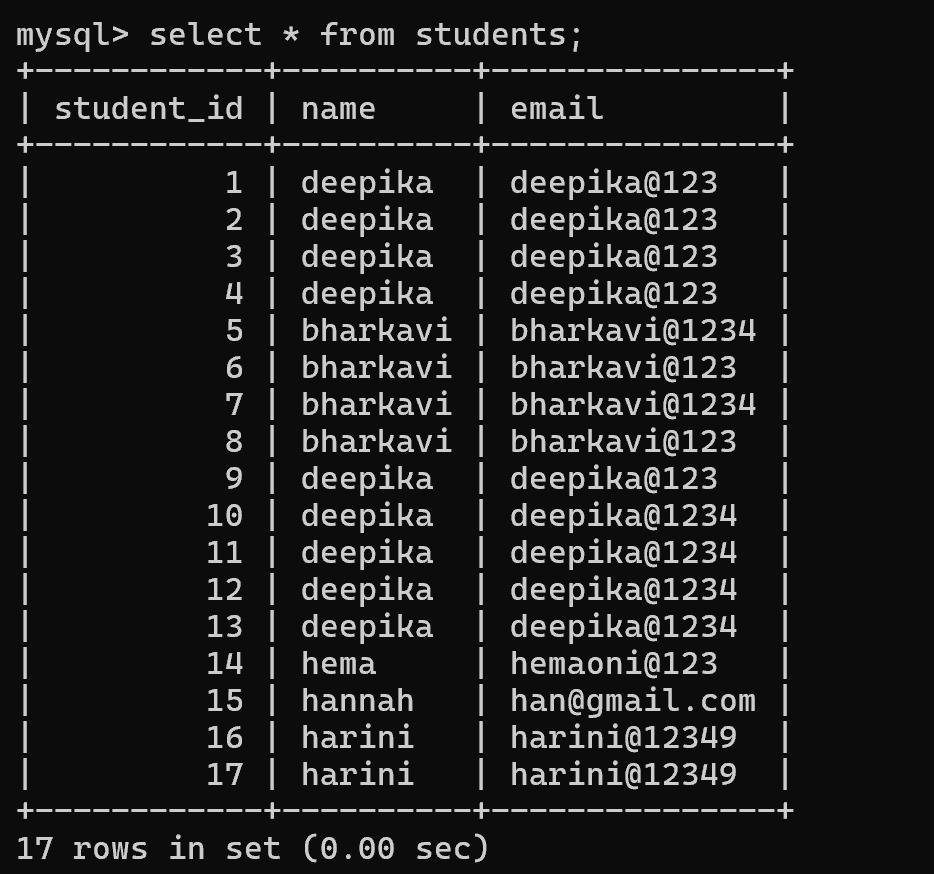
**Registration Table**





**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

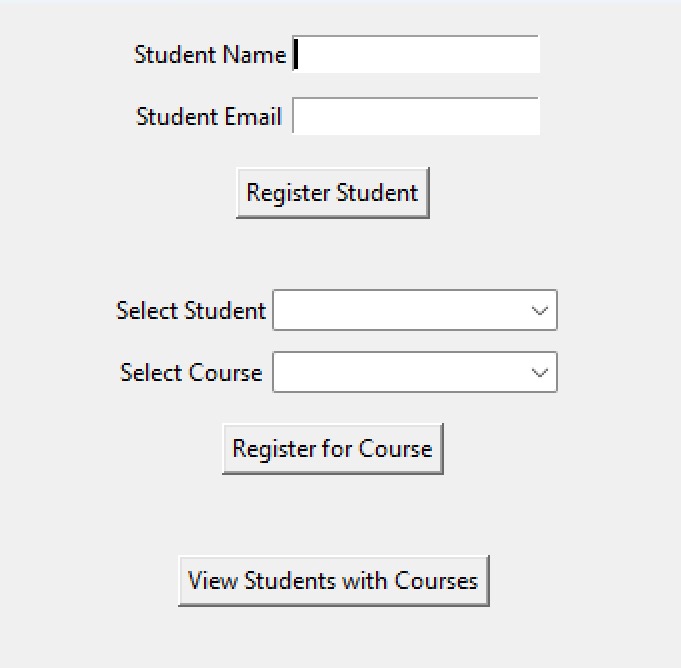
load\_data()

# Run the Tkinter event loop

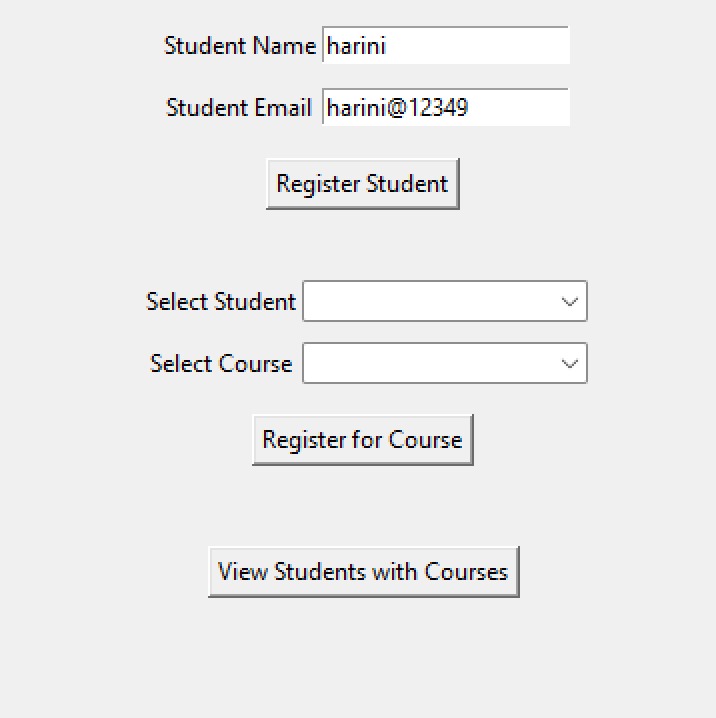
root.mainloop()

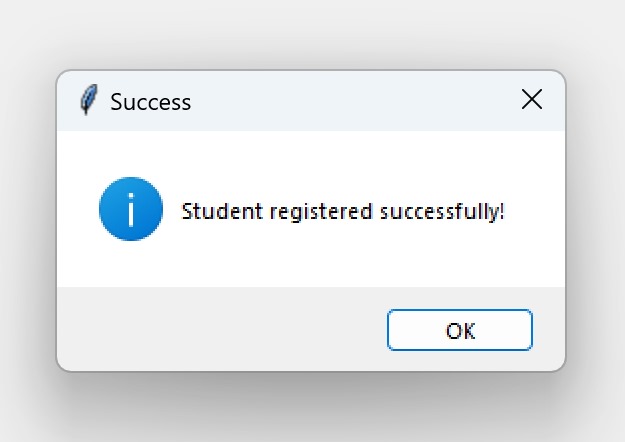
**V. RESULT AND DISCUSSION**

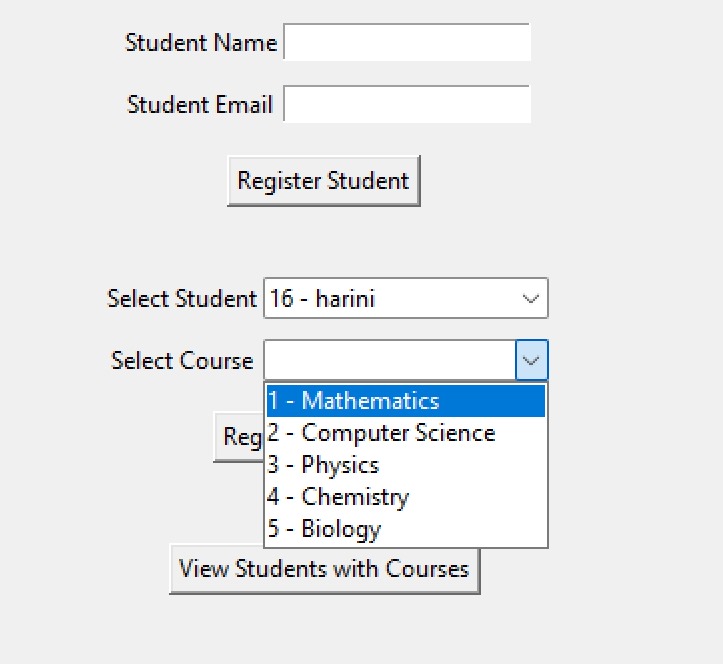
**LOGIN PAGE**

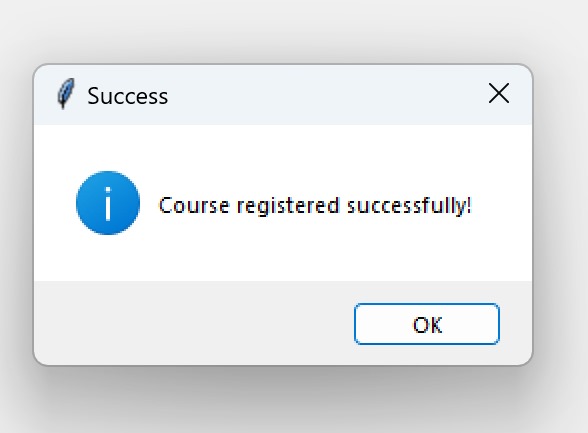


**REGISTRATION PAGE**

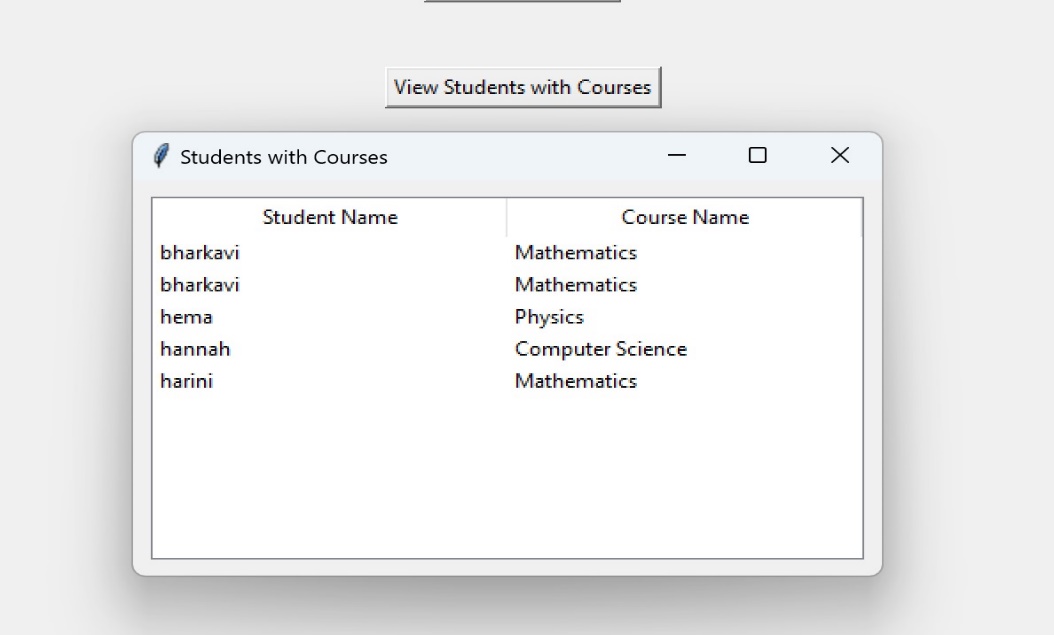








**CART PAGE**



**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>