

# STUDENT MANAGEMENT SYSTEM

A MINI PROJECT REPORT

**Submitted by**

RUDRA.S	220701231
SANJAY.S.M	220701249

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

THANDALAM

CHENNAI-602105

2023 - 24

## BONAFIDE CERTIFICATE

Certified that this project report **“STUDENT MANAGEMENT SYSTEM”** is the **bonafide work of “RUDRA(220701231),SANJAY(220701249) ”** 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-602105.

INTERNAL EXAMINER

EXTERNAL EXAMINER

## ABSTRACT

The provided Python code utilizes Tkinter to create a Student Management System GUI. It integrates with a MySQL database for student management.

The Student Management System (SMS) is a crucial tool for educational institutions to streamline and optimize various administrative tasks related to student information management. This abstract provides an overview of the features, benefits, and functionalities of a typical SMS.

The primary objective of an SMS is to efficiently manage student data throughout their academic journey. This includes storing personal information, academic records, attendance, grades, and other pertinent details in a centralized database. The system facilitates easy access to this information for authorized users such as administrators, faculty members, and students themselves.

# **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**

## **3. REQUIREMENTS AND ANALYSIS**

### **3.1 REQUIREMENT SPECIFICATION**

### **3.2 HARDWARE AND SOFTWARE REQUIREMENTS**

### **3.3 ARCHITECTURE DIAGRAM**

### **3.4 ER DIAGRAM**

### **3.5 NORMALIZATION**

## **4. PROGRAM CODE**

## **5. RESULTS AND DISCUSSION**

## **6. CONCLUSION**

# 1. Introduction

## 1.1 Introduction

A student management system is a comprehensive software solution designed to streamline and automate various administrative tasks within educational institutions. From managing student records and enrollment to tracking attendance, grades, and academic performance, these systems provide a centralized platform for educational institutions to efficiently manage student-related information. Additionally, they often include features for scheduling classes, communicating with students and parents, generating reports, and facilitating collaboration among teachers and staff. By digitizing and centralizing these processes, student management systems help educational institutions save time, reduce paperwork, improve data accuracy, and enhance overall efficiency in managing student affairs.

## 1.2 Objectives

Here are some objectives for a Student Management System (SMS):

1. **Efficient Student Information Management:** The primary objective of an SMS is to efficiently collect, organize, and manage student information in a centralized database. This includes personal details, academic records, attendance data, and other relevant information.
2. **Streamlined Enrollment Process:** The SMS aims to streamline the enrollment process by automating tasks such as course registration, fee payment, and class scheduling. This reduces manual efforts and minimizes errors, leading to a smoother experience for both students and administrators

3. **Enhanced Academic Performance Monitoring:** The system aims to facilitate better monitoring of students' academic performance by tracking their grades, attendance, and other relevant metrics. This enables educators to identify struggling students early on and provide timely interventions to support their success.
4. **Improved Communication Channels:** An SMS provides various communication tools such as messaging systems and portals to foster better communication between students, parents, faculty, and administrators. This facilitates timely exchange of information and enhances collaboration among stakeholders.
5. **Data-driven Decision Making:** By generating reports and analytics on student performance, attendance trends, and other metrics, the SMS enables administrators and educators to make informed, data-driven decisions. This helps in identifying areas for improvement and implementing targeted interventions to enhance student outcomes.
6. **Enhanced Administrative Efficiency:** One of the key objectives of an SMS is to improve administrative efficiency by automating routine tasks, reducing paperwork, and minimizing manual data entry. This frees up time and resources that can be redirected towards more strategic initiatives.
7. **Promotion of Student Engagement and Accountability:** Through features such as online portals and mobile applications, the SMS aims to promote student engagement and accountability by providing them with easy access to their academic information, course materials, and communication channels.

8. **Ensuring Data Security and Privacy:** An important objective of an SMS is to ensure the security and privacy of student data. This includes implementing robust security measures to protect against unauthorized access, data breaches, and other cyber threats.

9. **Integration with Other Systems:** The SMS aims to seamlessly integrate with other systems such as Learning Management Systems (LMS), finance systems, and HR systems to ensure interoperability and data consistency across the institution.

10. **Continuous Improvement and Adaptability:** Finally, the SMS should be designed to support continuous improvement and adaptability to evolving needs and technologies. This includes regularly updating the system based on user feedback, technological advancements, and changes in regulatory requirements.

### 1.3 Modules

The modules of a student management system typically include:

**Student Information Management:** This module manages student records, including personal details, contact information, academic history, and demographic data.

**Enrollment and Admission:** Handles the process of student enrollment, including application submission, admission criteria, and enrollment status tracking.

**Attendance Management:** Tracks student attendance, generates reports, and notifies stakeholders about attendance patterns.

**Academic Management:** Manages academic-related tasks such as course registration, scheduling, class assignments, and exam management.

**Grading and Transcript Management:** Records and calculates student grades, generates transcripts, and provides a comprehensive view of academic performance.

**Fee Management:** Manages student fees, including fee collection, payment tracking, fee waivers, and generating fee receipts.

**Communication Management:** Facilitates communication between students, parents, teachers, and administrators through features like messaging, announcements, and notifications.

**Library Management:** Handles library operations such as book circulation, cataloging, inventory management, and overdue book tracking.

**Hostel Management:** Manages hostel accommodations, room allocation, and facilities for students residing on campus.

**Transportation Management:** Manages transportation services for students, including bus routes, schedules, and tracking of transportation-related expenses.



**Extracurricular Activities Management:** Tracks participation in extracurricular activities, manages event schedules, and promotes student engagement outside of the classroom.

**Report Generation:** Generates various reports and analytics related to student performance, attendance, fees, and other relevant data for administrators, teachers, and parents.

These modules work together to streamline administrative processes, enhance communication, and improve overall efficiency in managing student affairs within educational institutions.

## 2. Survey of Technologies

### 2.1 Software Description

The software for the Student Management System is developed using Python, which is known for its simplicity and efficiency. The graphical user interface is created using Tkinter and CustomTkinter, libraries that are popular for developing desktop applications in Python. MySQL is employed as the database management system to store and manage data related to student details. Together, these technologies create a robust and userfriendly application for student management system.

1.     **Student Information Management:**
  - Capture and store detailed information about students, including personal details, contact information, academic history, and enrollment status.
  - Allow for easy retrieval and updating of student records.
2.     **Enrollment Management:**
  - Facilitate the enrollment process by managing admissions, registrations, course selections, and fee payments.
  - Generate enrollment reports and statistics for analysis.
3.     **Academic Records Management:**
  - Maintain comprehensive academic records for each student, including grades, attendance, class schedules, and academic performance evaluations.
  - Generate transcripts, progress reports, and other academic documents as required.
4.     **Course Management:**
  - Manage course offerings, curriculum details, class schedules, and faculty assignments.
  - Enable students to view available courses, register for classes, and track their academic progress.
5.     **Attendance Tracking:**

- Track student attendance automatically or manually for each class session.
- Generate attendance reports for students, faculty, and administration.
- 6. **Grading and Assessment:**
  - Record and calculate grades for assignments, quizzes, exams, and other assessments. ○ Allow faculty to input grades, provide feedback, and generate grade reports.
- 7. **Communication and Collaboration:**
  - Facilitate communication between students, faculty, and administrators through integrated messaging systems, announcements, and discussion forums. ○ Enable collaboration on assignments, projects, and group activities.
- 8. **Financial Management:**
  - Manage student fees, scholarships, financial aid, and payment processing.
  - Generate financial reports and statements for accounting purposes.
- 9. **Reporting and Analytics:**
  - Provide built-in reporting tools to generate various reports, such as student demographics, enrollment trends, academic performance, and financial summaries. ○ Offer analytics capabilities to identify patterns, trends, and areas for improvement.
- 10. **Security and Access Control:**
  - Ensure data security and privacy through role-based access control mechanisms. ○ Implement authentication and authorization protocols to restrict access to sensitive information.

**Benefits:**

- **Efficiency:** Streamline administrative processes and reduce manual paperwork, saving time and resources.
- **Accuracy:** Maintain accurate and up-to-date student records, grades, and academic information.

## 2.2 LANGUAGE USED

MYSQL (BACK END)

PYTHON(FRONT END)

TINKER

### MYSQL

MySQL is a popular open-source relational database management system (RDBMS) that is commonly used for building various types of applications, including student database management systems. Here's an overview of how MySQL can be utilized for a student database management system (DBMS):

**Data Modeling:** MySQL allows you to define the structure of your database using tables, columns, indexes, and relationships. For a student management system, you would typically define tables for students, courses, enrollment, faculty, attendance, grades, financials, and administration, as outlined in the normalization process.

**Data Storage and Retrieval:** MySQL provides efficient storage and retrieval mechanisms for managing large volumes of student data. It supports various data types (e.g., integer, varchar, date) and indexing options to optimize query performance.

**Data Integrity and Constraints:** MySQL allows you to enforce data integrity through constraints such as primary keys, foreign keys, unique constraints, and check constraints. This ensures that the data stored in the database remains consistent and accurate.

**Query Language:** MySQL uses Structured Query Language (SQL) for querying and manipulating data. You can use SQL statements to retrieve, insert, update, and delete data from the database tables. This flexibility

allows you to perform complex operations on student data, such as generating reports, calculating grades, and analyzing enrollment trends.

**Security:** MySQL provides robust security features to protect student data from unauthorized access and manipulation. You can create user accounts with specific privileges and roles, implement encryption for sensitive data, and configure access controls to restrict access to the database.

**Scalability and Performance:** MySQL is highly scalable and can handle large volumes of data and concurrent users. You can optimize database performance by fine-tuning configuration settings, indexing frequently queried columns, and using caching mechanisms.

**Integration and Compatibility:** MySQL integrates seamlessly with various programming languages, frameworks, and tools, making it easy to develop and deploy student management applications. It is compatible with popular web development technologies such as PHP, Python, Java, and Node.js.

**Community Support and Documentation:** MySQL has a large and active community of developers, users, and contributors who provide support, documentation, tutorials, and resources. You can find plenty of online forums, user groups, and documentation to help you with your student database management project.

Overall, MySQL is a reliable and feature-rich database management system that can serve as the backend for building robust and scalable student management applications. Whether you're managing student records, course information, attendance tracking, or financial transactions, MySQL provides the tools and capabilities you need to effectively manage student data.

## PYTHON

### . Backend Development:

- **Database Connectivity:** Python interacts with the database management system (e.g., MySQL, SQLite) to perform CRUD (Create, Read, Update, Delete) operations on student data. Libraries

like SQLAlchemy or Flask-MySQLdb facilitate database connectivity and query execution.

- **Business Logic:** Python implements the core functionalities of the SMS, such as handling user authentication, processing enrollment requests, calculating grades, and managing financial transactions. Object-oriented programming principles are often employed to structure the business logic into reusable and maintainable components.
- **API Development:** Python is used to develop RESTful APIs that expose endpoints for frontend interactions. Frameworks like Flask or Django simplify API development by providing routing mechanisms, request handling, and serialization/deserialization of data.

## 2. Frontend Development:

- **Web Frameworks:** Python web frameworks such as Flask or Django are used to develop the frontend components of the SMS. These frameworks provide templating engines (Jinja2 for Flask, Django Templates for Django) to generate dynamic HTML content based on server-side data.
- **User Interface:** Python, along with HTML, CSS, and JavaScript, is utilized to create the user interface of the SMS. Frontend frameworks like Bootstrap or Materialize CSS are often integrated to ensure responsive and visually appealing designs.
- **Form Handling:** Python is responsible for processing form submissions from users, validating input data, and triggering backend actions accordingly. Libraries like WTForms in Flask or Django Forms in Django simplify form handling and validation.

## 3. Automation and Scripting:

- **Data Import/Export:** Python scripts are employed to automate the import/export of student data to/from external sources (e.g., CSV files, Excel spreadsheets). Libraries like pandas facilitate data manipulation and transformation tasks.
- **Scheduled Tasks:** Python's `cron` or `schedule` libraries are utilized to schedule recurring tasks within the SMS, such as generating

reports, sending automated notifications, or performing database maintenance.

#### **4. Testing and Quality Assurance:**

- **Unit Testing:** Python's built-in `unittest` framework or third-party libraries like `pytest` are used to write and execute unit tests for individual components of the SMS, ensuring code reliability and correctness.
- **Integration Testing:** Python-based testing frameworks (e.g., Selenium for web application testing) are employed to perform integration tests that validate the interaction between different modules of the SMS.



## 3.REQUIREMENT ANALYSIS

### 3.1 Requirement Analysis

#### Functional Requirements

Functional requirements for a student management system outline the specific features and capabilities that the system must possess to effectively manage student-related tasks. Here are some essential functional requirements:

**Student Registration and Profile Management:** The system should allow administrators to register new students, update student information, and maintain comprehensive student profiles including personal details, contact information, and academic history.

**Enrollment and Admission:** Enable the process of student enrollment by managing applications, admission criteria, enrollment status, and document verification. Provide features for assigning students to courses and academic programs.

**Attendance Tracking:** Automate attendance tracking for classes, events, and activities. Provide options for teachers to record attendance electronically, view attendance reports, and notify stakeholders about absent students.

**Academic Management:** Facilitate course management, including scheduling, assignment submission, and exam management. Enable teachers to create course materials, manage class rosters, and track student progress.

**Grading and Transcript Management:** Support the recording and calculation of student grades, generate transcripts, and provide grade reports to students, parents, and administrators. Allow for grading customization and integration with grading scales.

**Fee Management:** Manage student fees, including fee collection, payment tracking, and fee waivers. Generate invoices, receipts, and financial reports related to student fees and payments.

**Communication Tools:** Provide communication features such as messaging, announcements, notifications, and discussion forums to facilitate communication between students, teachers, parents, and administrators. Allow for personalized communication and group messaging.

**Report Generation:** Generate various reports and analytics related to student performance, attendance, fees, and other relevant data. Provide customizable report templates and options for exporting reports in different formats.

**Integration with Learning Management Systems:** Integrate with learning management systems (LMS) to synchronize course materials, assignments, grades, and other academic data between the student management system and the LMS.

**Document Management:** Manage student-related documents such as admission forms, transcripts, certificates, and other records. Provide secure storage, retrieval, and sharing options for documents.

**User Roles and Permissions:** Define user roles and permissions to control access to system features and data. Assign different roles to administrators, teachers, students, and parents based on their responsibilities and permissions.

**Mobile Accessibility:** Ensure that the system is accessible via mobile devices, with responsive design and mobile apps available for administrators, teachers, students, and parents to access essential features and information on the go.

These functional requirements form the foundation of a robust student management system that effectively meets the needs of educational institutions and enhances efficiency in managing student affairs.

## **Non Functional Requirements**

Non-functional requirements for a student management system define the quality attributes, constraints, and characteristics that are essential for the system's performance, usability, security, and scalability. Here are some key non-functional requirements:

**Performance:** The system should be able to handle a large volume of concurrent users and transactions without significant performance degradation. Response times for critical functions such as user authentication, data retrieval, and report generation should be within acceptable limits.

**Scalability:** The system should be scalable to accommodate growth in student population and data volume over time. It should be able to handle increases in user load and system resources without impacting performance or reliability.

**Reliability:** The system should be highly reliable, with minimal downtime and data loss. It should have built-in redundancy, fault tolerance mechanisms, and backup procedures to ensure continuous availability and data integrity.

**Security:** The system should adhere to industry best practices for data security and privacy. It should implement robust authentication mechanisms, access controls, encryption, and audit trails to protect sensitive student information from unauthorized access, tampering, or disclosure.

**Usability:** The system should be user-friendly and intuitive, with a well designed interface that is easy to navigate and understand. It should support multiple languages, accessibility features, and customizable preferences to accommodate diverse user needs and preferences.

**Compatibility:** The system should be compatible with a variety of devices, operating systems, web browsers, and network environments. It should

support interoperability with other software applications and standards to facilitate data exchange and integration.

**Maintainability**: The system should be easy to maintain and upgrade, with modular architecture, clear documentation, and version control mechanisms. It should support automated testing, deployment, and monitoring to detect and address issues proactively.

**Compliance**: The system should comply with relevant laws, regulations, and industry standards related to data protection, privacy, accessibility, and security. This includes compliance with GDPR, FERPA, HIPAA, and other applicable regulations.

**Performance Monitoring**: The system should include monitoring tools and dashboards to track system performance, usage patterns, and resource utilization. It should provide alerts and notifications for performance bottlenecks, security incidents, and other critical events.

**Backup and Disaster Recovery**: The system should have robust backup and disaster recovery procedures in place to protect against data loss, corruption, or system failures. It should regularly backup data, perform disaster recovery drills, and have contingency plans for various scenarios.

By addressing these non-functional requirements, the student management system can ensure high performance, reliability, security, and usability, ultimately enhancing the overall user experience and satisfaction.

### **3.2 HARDWARE AND SOFTWARE REQUIREMENTS**

#### **HARDWARE SPECIFICATION:**

PROCESSOR : INTEL i3

MEMORY SIZE : 4GB

HDD : 256GB

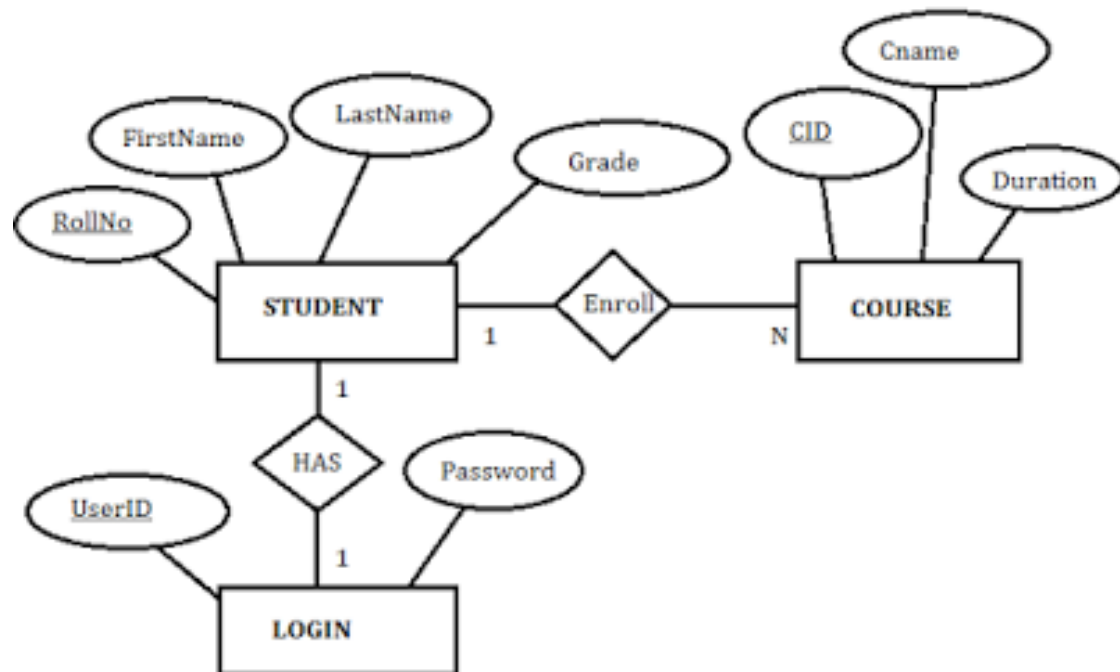
#### **SOFTWARE SPECTFICATION:**

OPERATING SYSTEM : WINDOWS 11

GUI INTERFACE : PYTHON

BACKEND : MY SQL

### 3.3 ER DIAGRAM



### **3.4 NORMALIZATION STEP**

#### **Normalization Steps:**

##### **First Normal Form (1NF):**

- Ensure that each table has a primary key.
- Eliminate repeating groups and ensure atomicity.
- Example: Break down a student's address into separate fields (street, city, state, zip).

##### **Second Normal Form (2NF):**

Meet the requirements of 1NF.

- Remove partial dependencies by separating subsets of data that apply to multiple records.
- Example: In the Enrollment table, separate Course details from Enrollment details.

##### **Third Normal Form (3NF):**

Meet the requirements of 2NF.

- Eliminate transitive dependencies by removing fields that are dependent on non-key attributes.
- Example: In the Grades table, move faculty information to a separate Faculty table.

## 4.PROGRAM CODE

```
#import libraries from
tkinter import *
import tkinter.ttk as ttk
import tkinter.messagebox as tkMessageBox
import sqlite3

#function to define database def
Database():
    global conn, cursor    #creating
    student database    conn =
    sqlite3.connect("student.db")    cursor
    = conn.cursor()

    #creating STUD_REGISTRATION table
    cursor.execute(
        "CREATE TABLE IF NOT EXISTS STUD_REGISTRATION (STU_ID INTEGER
        PRIMARY KEY AUTOINCREMENT NOT NULL, STU_NAME TEXT,
        STU_CONTACT TEXT, STU_EMAIL TEXT, STU_ROLLNO TEXT, STU_BRANCH
        TEXT)")

#defining function for creating GUI Layout
def DisplayForm():    #creating window
    display_screen = Tk()
```



```
#setting width and height for window
display_screen.geometry("900x400")

#setting title for window
display_screen.title("Student Management System")

global tree    global SEARCH

global name,contact,email,rollno,branch
SEARCH = StringVar()    name = StringVar()
contact = StringVar()    email = StringVar()
rollno = StringVar()    branch = StringVar()

#creating frames for layout

#topview frame for heading
TopViewForm = Frame(display_screen, width=600, bd=1, relief=SOLID)
TopViewForm.pack(side=TOP, fill=X)

#first left frame for registration from
LFrom = Frame(display_screen, width="350")
LFrom.pack(side=LEFT, fill=Y)

#seconf left frame for search form
LeftViewForm = Frame(display_screen, width=500,bg="gray")
LeftViewForm.pack(side=LEFT, fill=Y)

#mid frame for displaying students record
MidViewForm = Frame(display_screen, width=600)
MidViewForm.pack(side=RIGHT)

#label for heading
```

```
lbl_text = Label(TopViewForm, text="Student Management System",
font=('verdana', 18), width=600,bg="#1C2833",fg="white")
lbl_text.pack(fill=X)
```

```
#creating registration form in first left frame
```

```
Label(LFrom, text="Name ", font=("Arial", 12)).pack(side=TOP)
```

```
Entry(LFrom,font=("Arial",10,"bold"),textvariable=name).pack(side=TOP,
padx=10, fill=X)
```

```
Label(LFrom, text="Contact ", font=("Arial", 12)).pack(side=TOP)
```

```
Entry(LFrom, font=("Arial", 10,
"bold"),textvariable=contact).pack(side=TOP, padx=10, fill=X)
```

```
Label(LFrom, text="Email ", font=("Arial", 12)).pack(side=TOP)
```

```
Entry(LFrom, font=("Arial", 10,
"bold"),textvariable=email).pack(side=TOP, padx=10, fill=X)
```

```
Label(LFrom, text="Rollno ", font=("Arial", 12)).pack(side=TOP)
```

```
Entry(LFrom, font=("Arial", 10,
"bold"),textvariable=rollno).pack(side=TOP, padx=10, fill=X)
```

```
Label(LFrom, text="Branch ", font=("Arial", 12)).pack(side=TOP)
```

```
Entry(LFrom, font=("Arial", 10,
"bold"),textvariable=branch).pack(side=TOP, padx=10, fill=X)
```

```
Button(LFrom,text="Submit",font=("Arial", 10,
"bold"),command=register).pack(side=TOP, padx=10,pady=5, fill=X)
```

```
#creating search label and entry in second frame
```

```
lbl_txtsearch = Label(LeftViewForm, text="Enter name to Search",
font=('verdana', 10),bg="gray") lbl_txtsearch.pack() #creating
search entry
```

```
search = Entry(LeftViewForm, textvariable=SEARCH, font=('verdana',
15), width=10)
```

```
search.pack(side=TOP, padx=10, fill=X)

#creating search button

btn_search = Button(LeftViewForm, text="Search",
command=SearchRecord)

btn_search.pack(side=TOP, padx=10, pady=10, fill=X)

#creating view button

btn_view = Button(LeftViewForm, text="View All",
command=DisplayData)

btn_view.pack(side=TOP, padx=10, pady=10, fill=X)

#creating reset button

btn_reset = Button(LeftViewForm, text="Reset", command=Reset)
btn_reset.pack(side=TOP, padx=10, pady=10, fill=X)

#creating delete button

btn_delete = Button(LeftViewForm, text="Delete", command=Delete)
btn_delete.pack(side=TOP, padx=10, pady=10, fill=X)

#setting scrollbar

scrollbarx = Scrollbar(MidViewForm, orient=HORIZONTAL)
scrollbary = Scrollbar(MidViewForm, orient=VERTICAL)

tree = ttk.Treeview(MidViewForm,columns=("Student Id", "Name",
"Contact", "Email","Rollno","Branch"),
                    selectmode="extended", height=100,
yscrollcommand=scrollbary.set, xscrollcommand=scrollbarx.set)
scrollbary.config(command=tree.yview)
scrollbary.pack(side=RIGHT, fill=Y)
scrollbarx.config(command=tree.xview)
```

```

scrollbarx.pack(side=BOTTOM, fill=X)    #setting headings for
the columns

    tree.heading('Student Id', text="Student Id", anchor=W)
tree.heading('Name', text="Name", anchor=W)
tree.heading('Contact', text="Contact", anchor=W)
tree.heading('Email', text="Email", anchor=W)    tree.heading('Rollno',
text="Rollno", anchor=W)    tree.heading('Branch', text="Branch",
anchor=W)

    #setting width of the columns

    tree.column('#0', stretch=NO, minwidth=0, width=0)
tree.column('#1', stretch=NO, minwidth=0, width=100)
tree.column('#2', stretch=NO, minwidth=0, width=150) tree.column('#3',
stretch=NO, minwidth=0, width=80)

    tree.column('#4', stretch=NO, minwidth=0, width=120)
tree.pack()

    DisplayData()

#function to insert data into database def
register():

    Database()

    #getting form data    name1=name.get()    con1=contact.get()
email1=email.get()    rol1=rollno.get()    branch1=branch.get()
#applying empty validation    if name1==" or con1=="or
email1==" or rol1=="or branch1=="":
tkMessageBox.showinfo("Warning", "fill the empty field!!!")
else:

```

```
#execute query

conn.execute('INSERT INTO STUD_REGISTRATION
(STU_NAME,STU_CONTACT,STU_EMAIL,STU_ROLLNO,STU_BRANCH) \
VALUES (?,?,?,?,?),(name1,con1,email1,rol1,branch1));

conn.commit()

tkMessageBox.showinfo("Message","Stored successfully")

#refresh table data

DisplayData()

conn.close()
```

```
def Reset():

    #clear current data from table

tree.delete(*tree.get_children())

    #refresh table data

    DisplayData()

    #clear search text

SEARCH.set("")

name.set("")

contact.set("")

email.set("")

rollno.set("")

branch.set("") def

Delete():
```

```

        #open database
        Database()    if not
        tree.selection():
            tkMessageBox.showwarning("Warning","Select data to delete")
        else:
            result = tkMessageBox.askquestion('Confirm', 'Are you sure you want
            to delete this record?',
                                            icon="warning")

            if result == 'yes':
                curlItem = tree.focus()
                contents = (tree.item(curlItem))
                selecteditem = contents['values']
                tree.delete(curlItem)

                cursor=conn.execute("DELETE FROM STUD_REGISTRATION WHERE
                STU_ID = %d" % selecteditem[0])          conn.commit()
                cursor.close()          conn.close()

#function to search data def
SearchRecord():
    #open database
    Database()

    #checking search text is empty or not
    if SEARCH.get() != "":

```

```

        #clearing current display data
tree.delete(*tree.get_children())    #select
query with where clause

        cursor=conn.execute("SELECT * FROM STUD_REGISTRATION WHERE
STU_NAME LIKE ?", ('%' + str(SEARCH.get()) + '%',))

        #fetch all matching records
fetch = cursor.fetchall() #loop for
displaying all records into GUI
for data in fetch:    tree.insert("",
'end', values=(data))
cursor.close()    conn.close()

#defining function to access data from SQLite database def
DisplayData():

    #open database

    Database()    #clear current data
tree.delete(*tree.get_children())

    #select query

    cursor=conn.execute("SELECT * FROM STUD_REGISTRATION")

    #fetch all data from database

fetch = cursor.fetchall()

    #loop for displaying all data in GUI
for data in fetch:    tree.insert("",
'end', values=(data))    cursor.close()
conn.close()

```

#calling function

DisplayForm() if

\_name=='main\_':

#Running

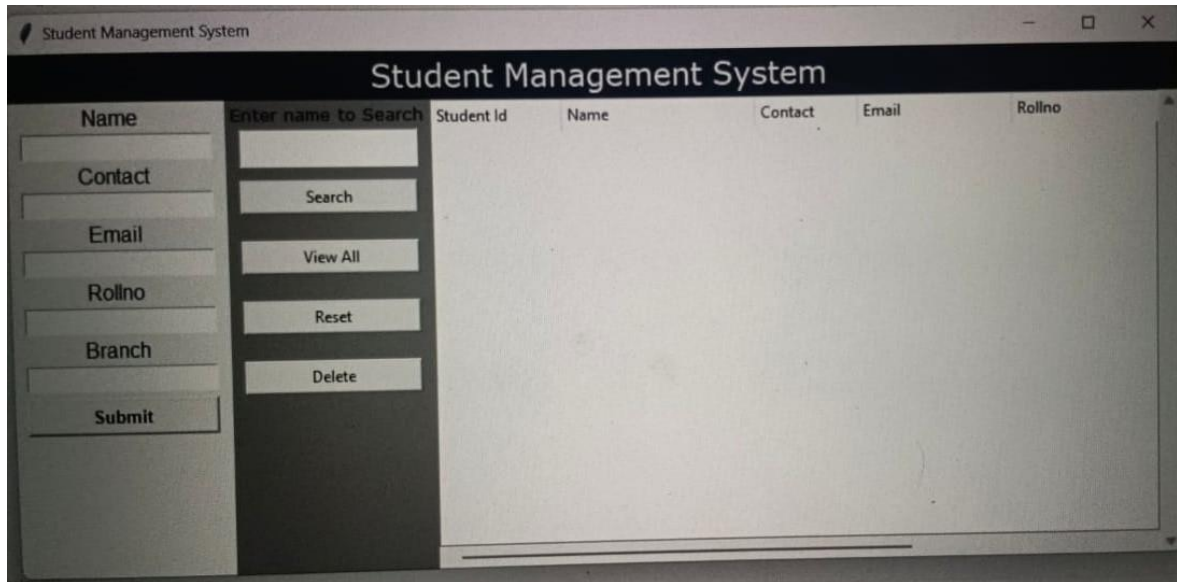
Application

mainloop()



## 5.RESULTS AND DISCUSSION

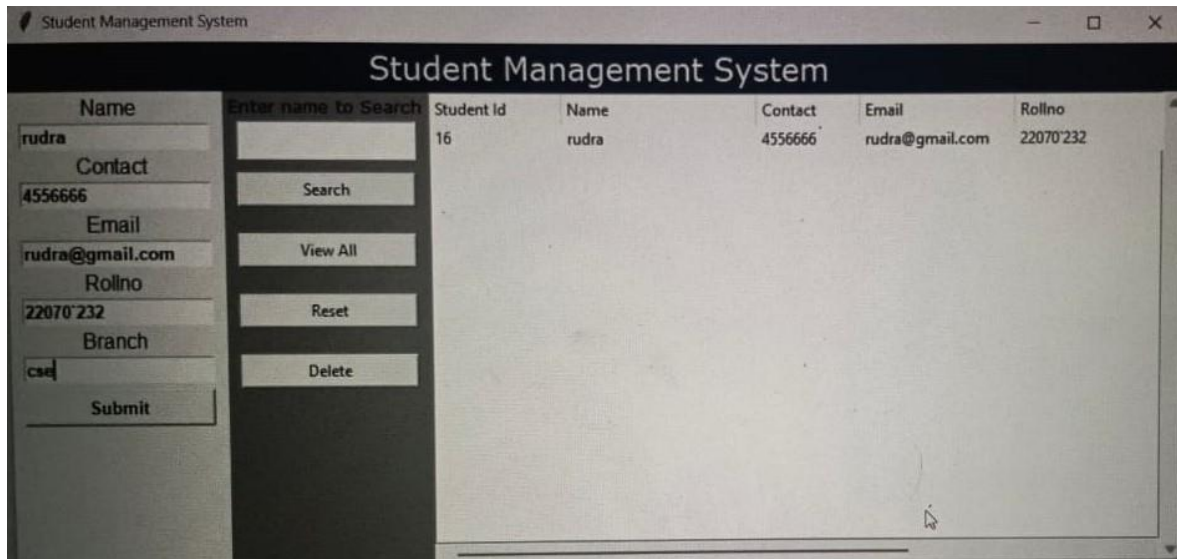
### HOME PAGE



The screenshot shows the home page of a 'Student Management System'. The interface includes a sidebar with input fields for Name, Contact, Email, Rollno, Branch, and a Submit button. The main area features a search bar labeled 'Enter name to Search' with buttons for Search, View All, Reset, and Delete. A table with columns for Student Id, Name, Contact, Email, and Rollno is present but currently empty.

Student Id	Name	Contact	Email	Rollno
------------	------	---------	-------	--------

### ENTERING THE STUDENT DETAILS



This screenshot shows the same system after a student's details have been entered. The sidebar fields are now populated with the student's information. The main table displays one record for a student named 'rudra'.

Student Id	Name	Contact	Email	Rollno
16	rudra	4556666	rudra@gmail.com	22070'232

## SEARCH FOR A STUDENT DETAILS

The screenshot shows the 'Student Management System' window. On the left, there is a form with fields for Name, Contact, Email, Rollno, and Branch, each with a corresponding input box. Below these fields is a 'Submit' button. In the center, there is a search section with a text input field labeled 'Enter name to Search' containing the text 'rudra'. Below this input field are four buttons: 'Search', 'View All', 'Reset', and 'Delete'. On the right, a table displays student details. The table has columns for Student Id, Name, Contact, Email, and Rollno. The first row shows the details for a student named 'rudra' with Student Id 16, Contact 4556666, Email rudra@gmail.com, and Rollno 22070'232.

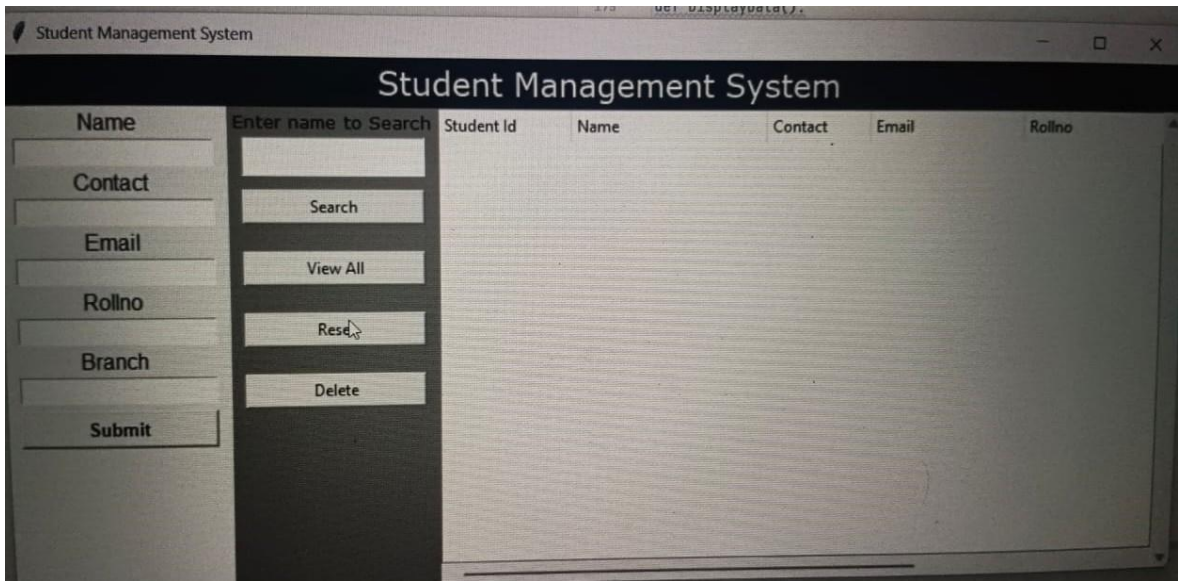
Student Id	Name	Contact	Email	Rollno
16	rudra	4556666	rudra@gmail.com	22070'232

## DELETE THE STUDENT RECORD

The screenshot shows the 'Student Management System' window. On the left, there is a form with fields for Name, Contact, Email, Rollno, and Branch, each with a corresponding input box. Below these fields is a 'Submit' button. In the center, there is a search section with a text input field labeled 'Enter name to Search' containing the text 'rudra'. Below this input field are four buttons: 'Search', 'View All', 'Reset', and 'Delete'. On the right, a table displays student details. The table has columns for Student Id, Name, Contact, Email, and Rollno. The first row shows the details for a student named 'rudra' with Student Id 16, Contact 4556666, Email rudra@gmail.com, and Rollno 22070'232. A confirmation dialog box is overlaid on the table, asking 'Are you sure you want to delete this record?' with 'Yes' and 'No' buttons.

Student Id	Name	Contact	Email	Rollno
16	rudra	4556666	rudra@gmail.com	22070'232

RESET THE STUDENT RECORD



## 6.CONCLUSION

In conclusion, a Student Management System (SMS) is an essential tool for educational institutions to efficiently manage student data, streamline administrative processes, and enhance communication between students, faculty, and administrators. Throughout this process, we've explored various aspects of designing and implementing an SMS, including:

1. **Requirements Analysis:** Understanding the specific needs and requirements of the educational institution to tailor the SMS accordingly.
2. **Software Description:** Outlining the key features and functionalities of the SMS, such as student information management, enrollment management, academic records management, course management, attendance tracking, grading and assessment, financial management, reporting and analytics, communication and collaboration, and security and access control.
3. **Normalization:** Designing a normalized database schema to organize student data efficiently, minimize redundancy, and maintain data integrity.
4. **Technology Stack:** Selecting appropriate technologies for building the SMS, such as MySQL for database management and Python with Flask or Django for web application development.
5. **Development:** Implementing the SMS features using chosen technologies, including creating database tables, developing frontend and backend components, and integrating security measures.
6. **Conclusion:** A well-designed and implemented Student Management System offers numerous benefits, including increased efficiency, accuracy, transparency, data-driven decision-making, scalability, and improved educational outcomes. It serves as a centralized platform for managing student information, academic records, enrollment processes, and communication channels within educational institutions.

