Faculty of Engineering & Technology			
Ramaiah University of Applied Sciences			
Department	Computer Science and Engineering	Programme	B. Tech. CSE/AIML/ISE
Semester/Batch	5th/2021		
Course Code	20CSC302A	**Course Title**	Database Systems
Course Leader(s)	Dr. Narendra Babu, Mrs. Sahana P Shankar, Mrs. Supriya M S		

Assignment						
Register No.	**21ETIS411028**	Name of Student	**R. KAARTHIGEYAN**			
		Marking Scheme	**Max Marks**	First Examiner Marks	**Second Examiner Marks**	
Part A	**A.1**	Discuss any two database models used in the modern day enterprise computing applications with suitable examples.	05			
		Part-A Max Marks	05			
Part B	B2.1	List of functional requirements	02			
	B2.2	Implementation of relational database schema with appropriate attributes, and constraints using SQL commands	10			
	B2.3	Design and implementation of GUI	05			
	B2.4	Connection of front end with the database and discussion on the results	03			
		Part-B Max Marks	20			
		Total Assignment Marks	25			

RAMAIAH UNIVERSITY OF APPLIED SCIENCES

Course Marks Tabulation				
Component-1(B)Assignment	First Examiner	Remarks	Second Examiner	Remarks
A				
B				
Marks (out of 25)				

Signature of First Examiner Signature of Second Examiner

Please note:

Documental evidence for all the components/parts of the assessment such as the reports, photographs, laboratory exam / tool tests are required to be attached to the assignment report in a proper order.
The First Examiner is required to mark the comments in RED ink and the Second Examiner's comments should be in GREEN ink.
The marks for all the questions of the assignment have to be written only in the **Component – CET B: Assignment** table.
If the variation between the marks awarded by the first examiner and the second examiner lies within +/- 3 marks, then the marks allotted by the first examiner is considered to be final. If the variation is more than +/- 3 marks then both the examiners should resolve the issue in consultation with the Chairman BoE.

Assignment
Instructions to students:
The assignment consists of **2** questions: Part A –**1** Question, Part B- **1** Questions.
Maximum marks is **25**.
The assignment has to be neatly word processed as per the prescribed format.
Submission Date: 12/03/2024
Submission after the due date is not permitted.
IMPORTANT: It is essential that all the sources used in preparation of the assignment must be suitably referenced in the text.
Marks will be awarded only to the sections and subsections clearly indicated as per the problem statement/exercise/question

PART A **05 Marks**

Enterprise computing applications are software applications designed to meet the complex and extensive requirements of large organizations or enterprises. These applications are critical for supporting various business processes, enhancing efficiency, and facilitating collaboration across different departments. Some the examples include Enterprise Resource Planning, Customer Relationship Management, Business Intelligence, Project Management Systems among others. You are required to generate a short report (no exceeding 300 Words) on the context which should address the following:

Discuss any two database models used in the modern day enterprise computing applications with suitable examples.

In modern enterprise computing applications, two prevalent database models are relational databases and NoSQL databases.

1. Relational Databases:

Relational databases organize data into tables consisting of rows and columns, with relationships established between tables through keys. They are characterized by their structured format and adherence to ACID (Atomicity, Consistency, Isolation, Durability) properties, ensuring data integrity and reliability. Examples of relational databases commonly used in enterprise applications include:

a. **MySQL**: Widely utilized for its reliability, scalability, and ease of use, MySQL powers numerous enterprise applications across various industries. It is an open-source relational database management system (RDBMS) renowned for its robust performance and comprehensive feature set.

b. **Oracle Database:** As one of the most established relational database management systems, Oracle Database is prevalent in large-scale enterprise environments. It offers advanced features such as partitioning, clustering, and high availability, making it suitable for mission-critical applications demanding high performance and scalability.

2. NoSQL Databases:

NoSQL databases diverge from the structured nature of relational databases, offering more flexibility and scalability for handling large volumes of unstructured or semi-structured data. They are particularly well-suited for use cases where traditional relational databases may encounter limitations. Examples include:

a. **MongoDB**: A leading document-oriented NoSQL database, MongoDB is widely adopted in enterprise applications for its scalability and agility. It stores data in flexible JSON-like documents, allowing for dynamic schema evolution and seamless horizontal scalability across distributed clusters.

b. **Apache Cassandra**: Recognized for its ability to handle massive amounts of data with high availability and fault tolerance, Cassandra is a distributed NoSQL database ideal for enterprise applications requiring robust performance and scalability. It excels in scenarios involving large-scale data storage and real-time analytics.

These database models cater to diverse enterprise needs, providing the foundation for robust and scalable data management solutions tailored to specific requirements and use cases. Whether it's the structured reliability of relational databases or the flexible scalability of NoSQL databases, enterprises can leverage these models to effectively manage and derive insights from their data assets.

REFERENCES:

Lahiri, T., Chavan, S., Colgan, M., Das, D., Ganesh, A., Gleeson, M., ... & Zait, M. (2015, April). Oracle database in-memory: A dual format in-memory database. In *2015 IEEE 31st International Conference on Data Engineering* (pp. 1253-1258). IEEE.
Strauch, C., Sites, U. L. S., & Kriha, W. (2011). NoSQL databases. *Lecture Notes, Stuttgart Media University*, *20*(24), 79.

PART B **20 Marks**

Consider the **RUAS Student Management System** to manage the details of students in RUAS. The computerized system enables the users to access students' data at any time and from any place. The system consists of the functionalities such as Student Details, Branch Details, Fee Payment, Exam Results and any other student related information needed by the university. It is required to undertake the following activities:

B2.1 List of functional requirements

Sure, here are the functional requirements in a simpler format:

1**. Student Registration:**
- Staff can add new students, entering their name, contact info, email, roll number, and branch. Each student gets a unique ID.

2. **Branch Management:**
- Staff can add, view, update, and delete branch details, including name, location, and contact number.

3. **Fee Payment Processing:**
- Staff can record student fee payments, including ID, payment date, and amount. Each payment gets a unique ID.

4**. Exam Result Management**:
- Staff can record exam results for students, including ID, exam date, and marks obtained. Each result gets a unique ID.

5. **Viewing Student Information**:
- Staff can easily view student details, including personal info, fee payment history, and exam results.

6**. Updating Student Information:**
- Staff can update student details like contact info or branch enrollment, with validation to ensure accuracy.

7. **Search Functionality**:
- Staff can search for specific student records using criteria like name, roll number, or branch.

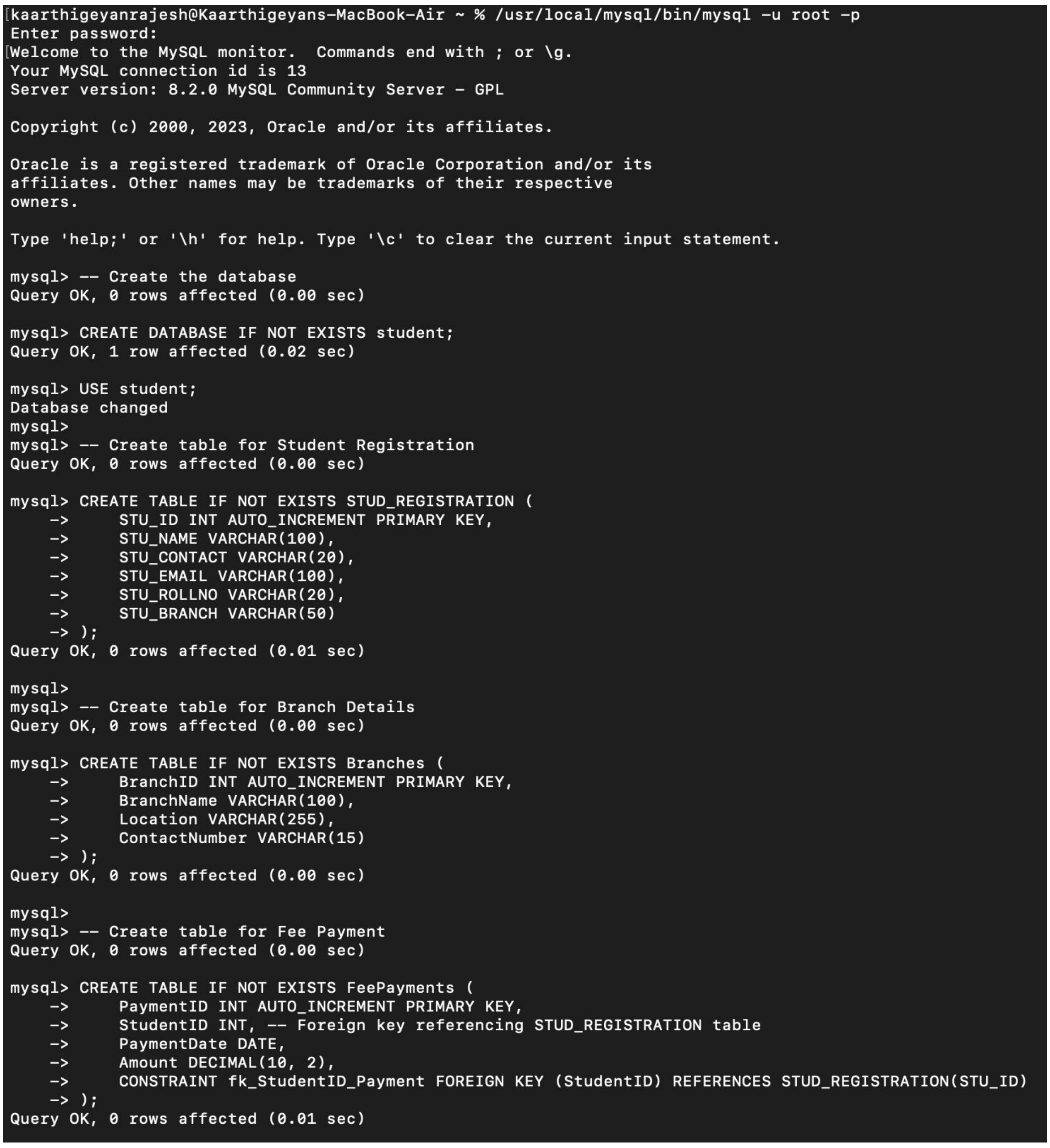
8. **Reporting:**
   - Staff can generate reports on fee payment status, exam results, or branch-wise enrollment, customizable and exportable.

9. **Security and Access Control:**
   - User authentication ensures only authorized staff can access and modify student information.

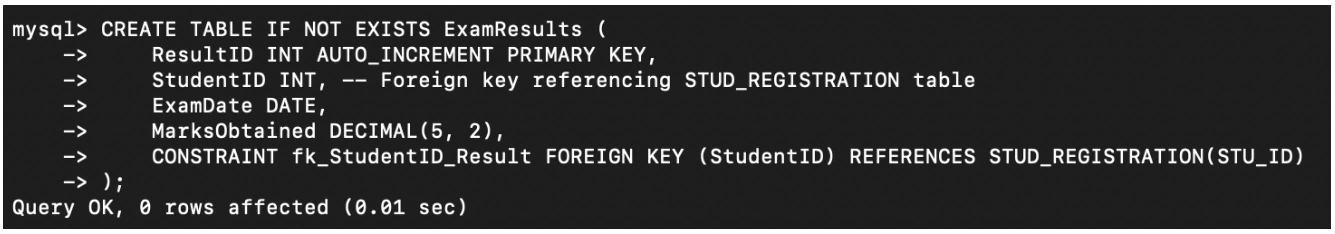
10. **Data Integrity and Validation:**
   - System enforces rules to ensure accurate and consistent data entry, handling errors effectively.

11. **Audit Trail:**
   - System keeps a record of all changes made to student records, accessible for accountability and tracking purposes.

B2.2 Implementation of relational database schema with appropriate attributes, and constraints using SQL commands



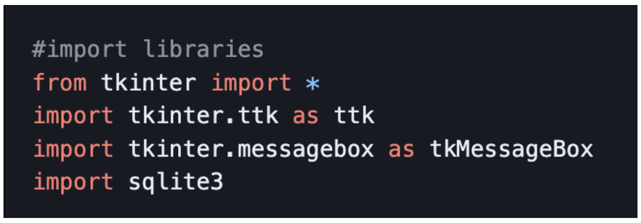
```
[kaarthigeyanrajesh@Kaarthigeyans-MacBook-Air ~ % /usr/local/mysql/bin/mysql -u root -p
Enter password:
[Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 13
Server version: 8.2.0 MySQL Community Server - GPL

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> -- Create the database
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE DATABASE IF NOT EXISTS student;
Query OK, 1 row affected (0.02 sec)

mysql> USE student;
Database changed
mysql>
mysql> -- Create table for Student Registration
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE IF NOT EXISTS STUD_REGISTRATION (
    ->     STU_ID INT AUTO_INCREMENT PRIMARY KEY,
    ->     STU_NAME VARCHAR(100),
    ->     STU_CONTACT VARCHAR(20),
    ->     STU_EMAIL VARCHAR(100),
    ->     STU_ROLLNO VARCHAR(20),
    ->     STU_BRANCH VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> -- Create table for Branch Details
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE IF NOT EXISTS Branches (
    ->     BranchID INT AUTO_INCREMENT PRIMARY KEY,
    ->     BranchName VARCHAR(100),
    ->     Location VARCHAR(255),
    ->     ContactNumber VARCHAR(15)
    -> );
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> -- Create table for Fee Payment
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE IF NOT EXISTS FeePayments (
    ->     PaymentID INT AUTO_INCREMENT PRIMARY KEY,
    ->     StudentID INT, -- Foreign key referencing STUD_REGISTRATION table
    ->     PaymentDate DATE,
    ->     Amount DECIMAL(10, 2),
    ->     CONSTRAINT fk_StudentID_Payment FOREIGN KEY (StudentID) REFERENCES STUD_REGISTRATION(STU_ID)
    -> );
Query OK, 0 rows affected (0.01 sec)
```



```
mysql> CREATE TABLE IF NOT EXISTS ExamResults (
    ->     ResultID INT AUTO_INCREMENT PRIMARY KEY,
    ->     StudentID INT, -- Foreign key referencing STUD_REGISTRATION table
    ->     ExamDate DATE,
    ->     MarksObtained DECIMAL(5, 2),
    ->     CONSTRAINT fk_StudentID_Result FOREIGN KEY (StudentID) REFERENCES STUD_REGISTRATION(STU_ID)
    -> );
Query OK, 0 rows affected (0.01 sec)
```

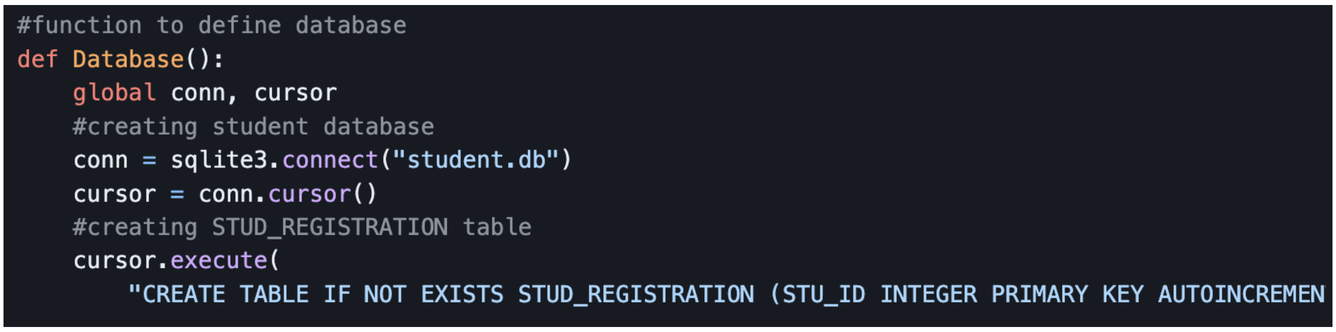
Design and implementation of GUI

1. **import library:** Here in this step we will import the needed library that will be used to create Student Management System. Here as you can see that there are three libraries are imported in which tkinter library is using to create a GUI window , tkinter.messagebox library is using to display message in Popup box and sqlite3 library is using to handle SQLite database.



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

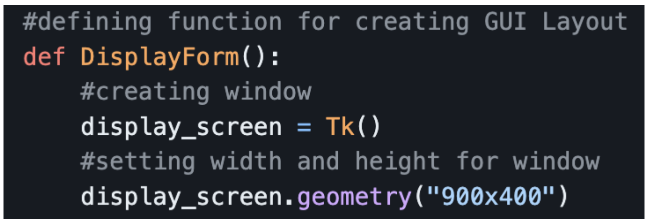
2. **Create Database and Table:** I have defined a function with name Database() to create database and table. Here is the following code to create database and table.



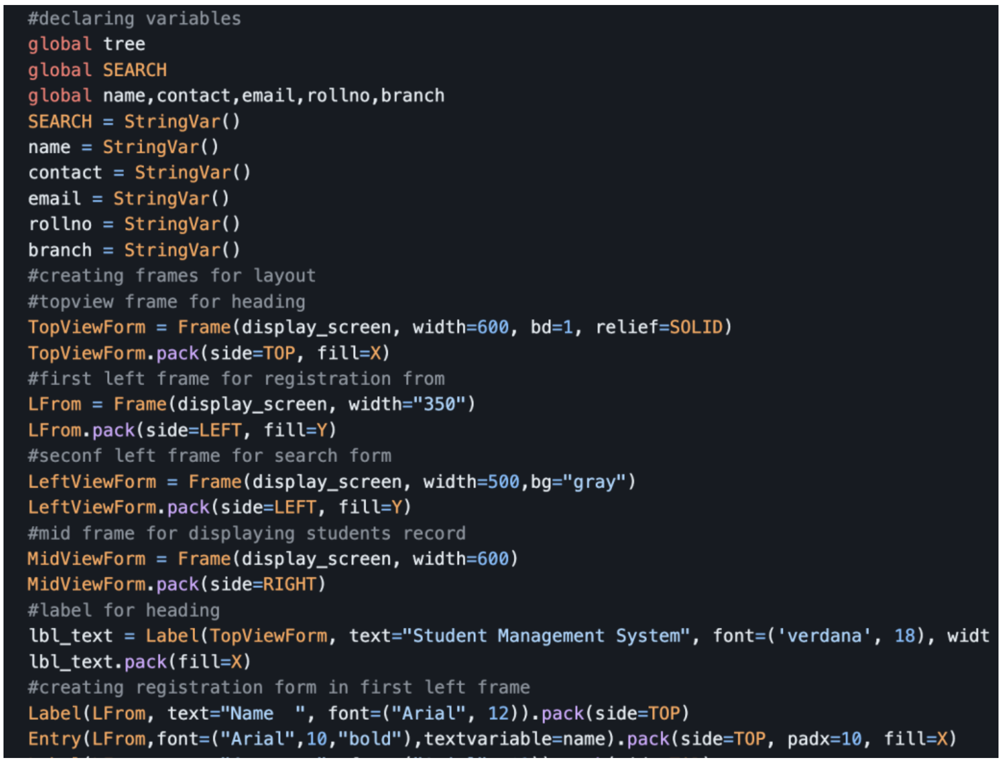
```python
#function to define database
def Database():
    global conn, cursor
    #creating student database
    conn = sqlite3.connect("student.db")
    cursor = conn.cursor()
    #creating STUD_REGISTRATION table
    cursor.execute(
        "CREATE TABLE IF NOT EXISTS STUD_REGISTRATION (STU_ID INTEGER PRIMARY KEY AUTOINCREMEN
```

Here in the above program database name is student and table name is STUD_REGISTRATION. The fields of table STUD_REGISTRATION are STU_ID , STU_NAME, STU_CONTACT, STU_EMAIL, STU_ROLLNO AND STU_BRANCH

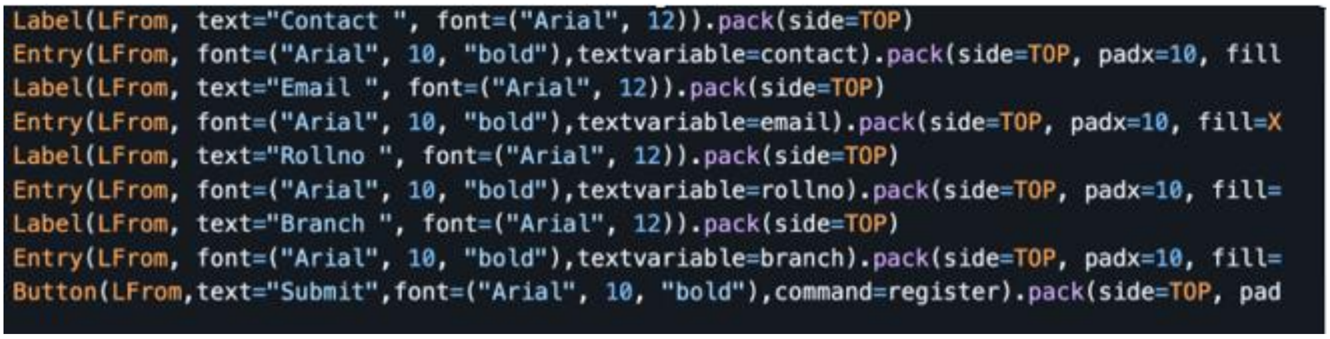
3. **Create GUI Window** :I have used four frame to create this Graphical user interface form in which first frame for Heading, second frame for Registration form, third frame for search window and the last frame for displaying students records. Here is the complete code to create GUI of this application.



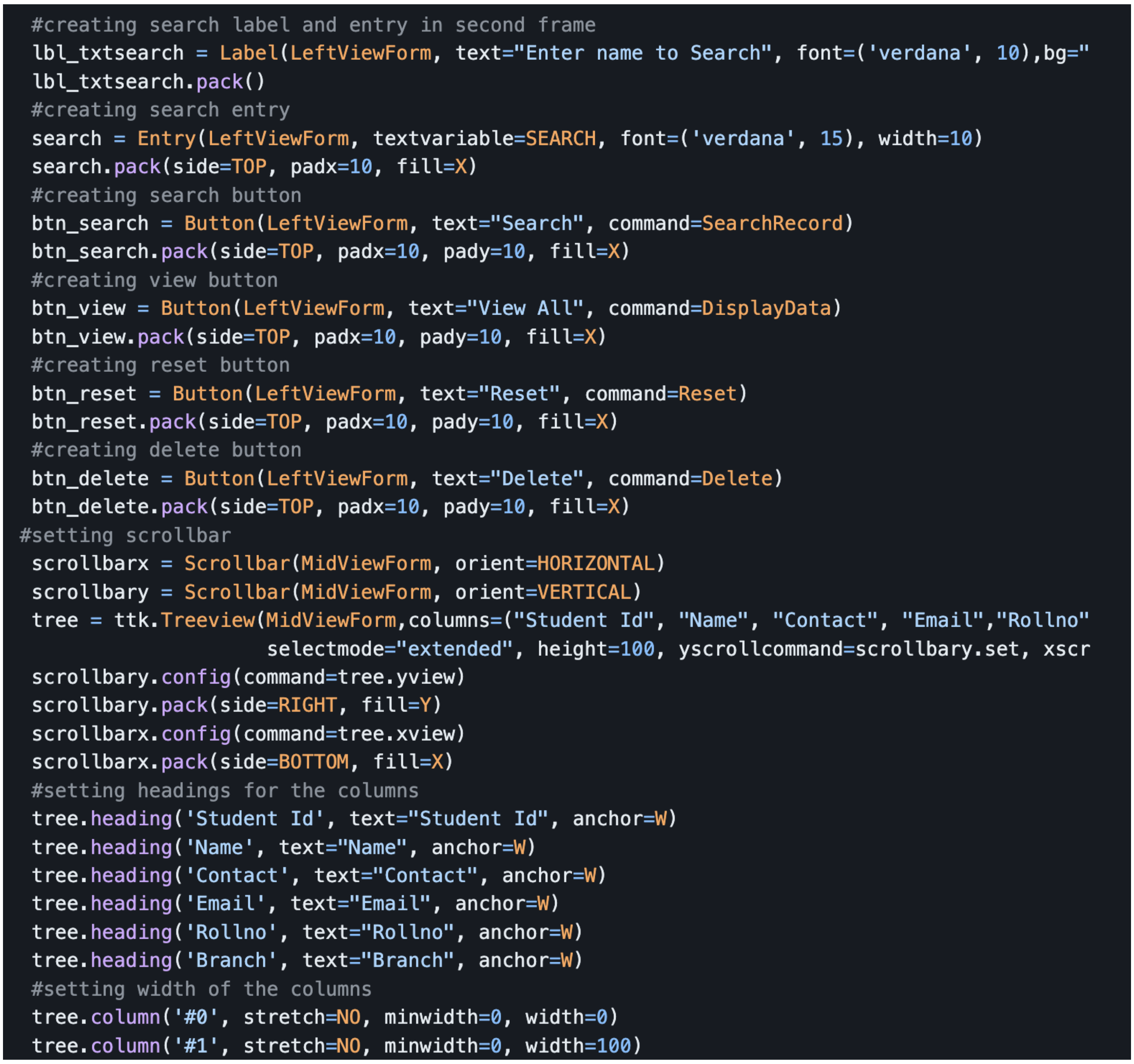
```
#defining function for creating GUI Layout
def DisplayForm():
    #creating window
    display_screen = Tk()
    #setting width and height for window
    display_screen.geometry("900x400")
```



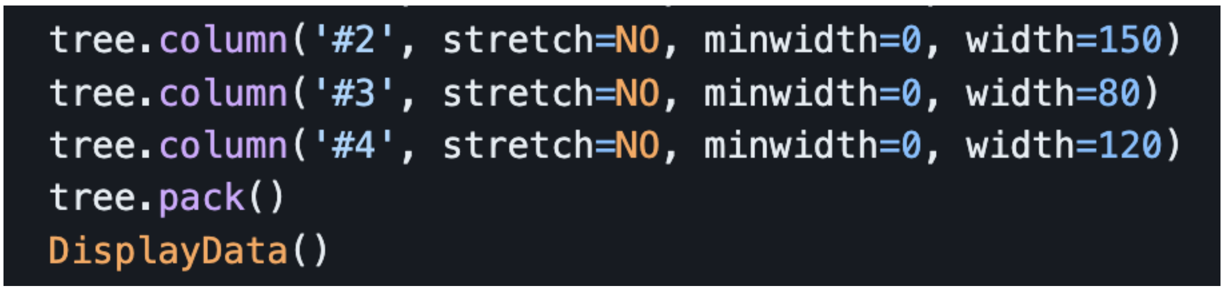
```
    #declaring variables
    global tree
    global SEARCH
    global name,contact,email,rollno,branch
    SEARCH = StringVar()
    name = StringVar()
    contact = StringVar()
    email = StringVar()
    rollno = StringVar()
    branch = StringVar()
    #creating frames for layout
    #topview frame 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), widt
    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
    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=
    Label(LFrom, text="Branch ", font=("Arial", 12)).pack(side=TOP)
    Entry(LFrom, font=("Arial", 10, "bold"),textvariable=branch).pack(side=TOP, padx=10, fill=
    Button(LFrom,text="Submit",font=("Arial", 10, "bold"),command=register).pack(side=TOP, pad
```

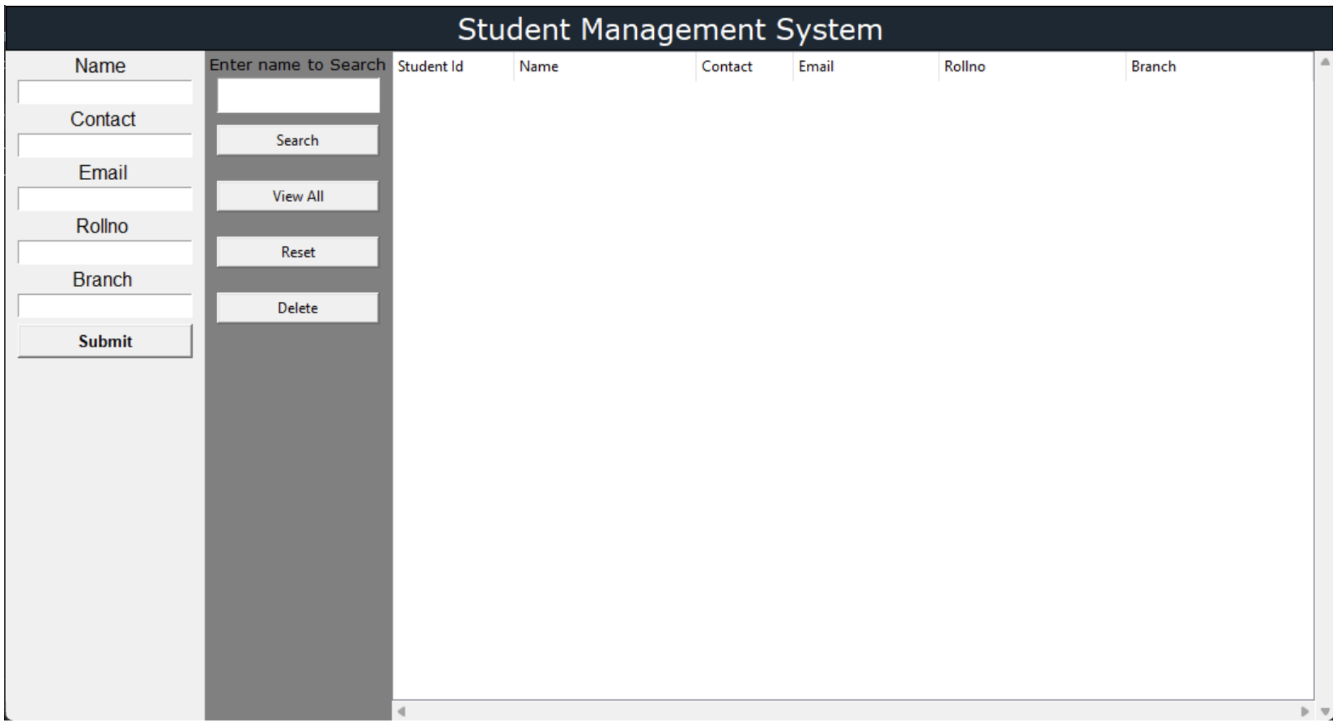


```
 #creating search label and entry in second frame
 lbl_txtsearch = Label(LeftViewForm, text="Enter name to Search", font=('verdana', 10),bg="
 lbl_txtsearch.pack()
 #creating search entry
 search = Entry(LeftViewForm, textvariable=SEARCH, font=('verdana', 15), width=10)
 search.pack(side=TOP, padx=10, 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"
                     selectmode="extended", height=100, yscrollcommand=scrollbary.set, xscr
 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()
```

THE GUI WINDOW :



B2.4. Connection of front end with the database and discussion on the results.

import library: Here in this step we will import the needed library that will be used to create Student Management system. Also here, tkinter.messagebox displays message in Pop up box.

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

Create Database and Table: I have defined a function with name Database() to create database and table. Here is the following code to create database and table.

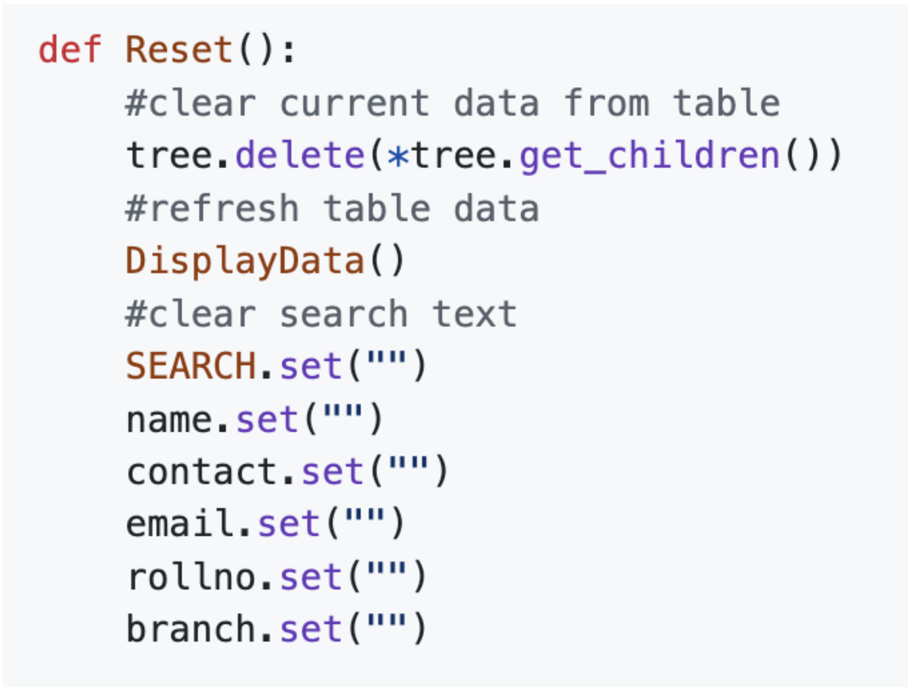
```
#function to define database
def Database():
    global conn, cursor
    #creating student database
    conn = sqlite3.connect("student.db")
    cursor = conn.cursor()
    #creating STUD_REGISTRATION table
    cursor.execute(
        "CREATE TABLE IF NOT EXISTS STUD_REGISTRATION (STU_ID INTEGER PRIMARY KEY AUTOINCREMEN
```

Here in the above program database name is student and table name is STUD_REGISTRATION. The fields of table STUD_REGISTRATION are STU_ID , STU_NAME, STU_CONTACT, STU_EMAIL, STU_ROLLNO AND STU_BRANCH

Insert Data into table: I have defined a function with name register(). First it will open database by calling Database() function. Next i have passed all the forms data into Python variable. After that i have applied a empty validation and finally at last i have applied query to insert data into table Then displayed data screen by calling DisplayData() function. Here is the complete code to insert data into table.

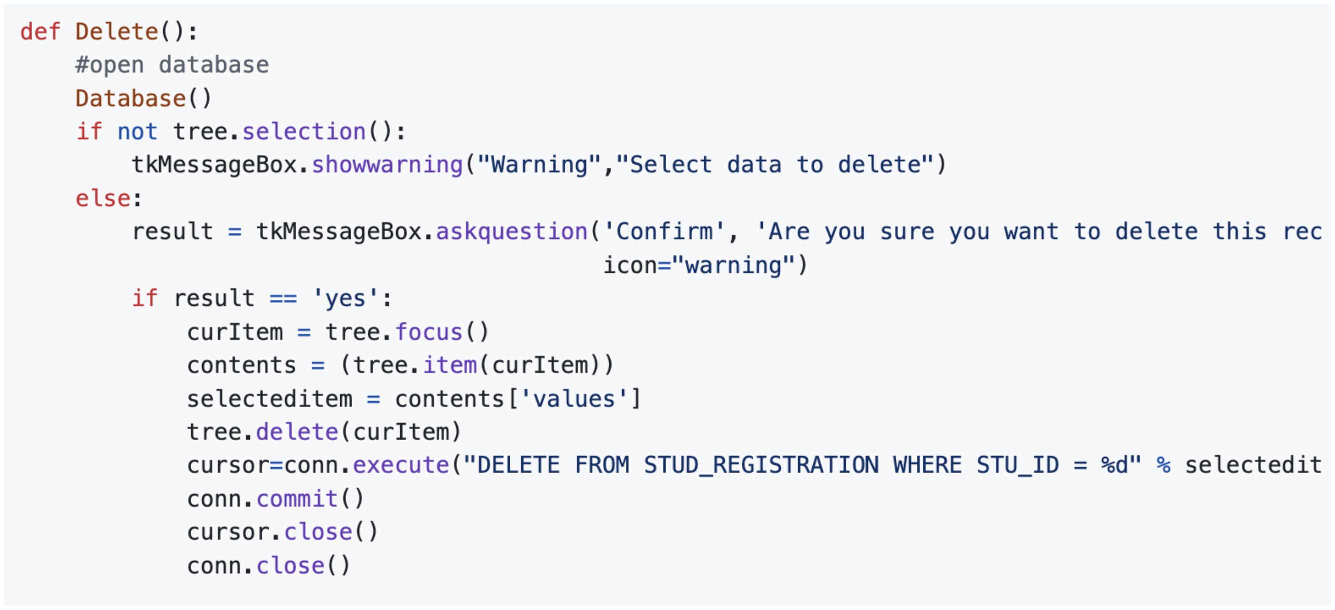
```
#function to insert data into database
def register():
    Database()
    #getting form data
    name1=name.get()
    con1=contact.get()
    email1=email.get()
    rol1=rollno.get()
    branch1=branch.get()
    #applying empty validation
    if name1=='' or con1==''or email1=='' or rol1==''or branch1=='':
        tkMessageBox.showinfo("Warning","fill the empty field!!!")
    else:
        #execute query
        conn.execute('INSERT INTO STUD_REGISTRATION (STU_NAME,STU_CONTACT,STU_EMAIL,STU_ROLLNO
 VALUES (?,?,?,?,?)',(name1,con1,email1,rol1,branch1));
        conn.commit()
        tkMessageBox.showinfo("Message","Stored successfully")
        #refresh table data
        DisplayData()
        conn.close()
```

Reset Form: I have defined a function Reset() and it will reset all the from data.



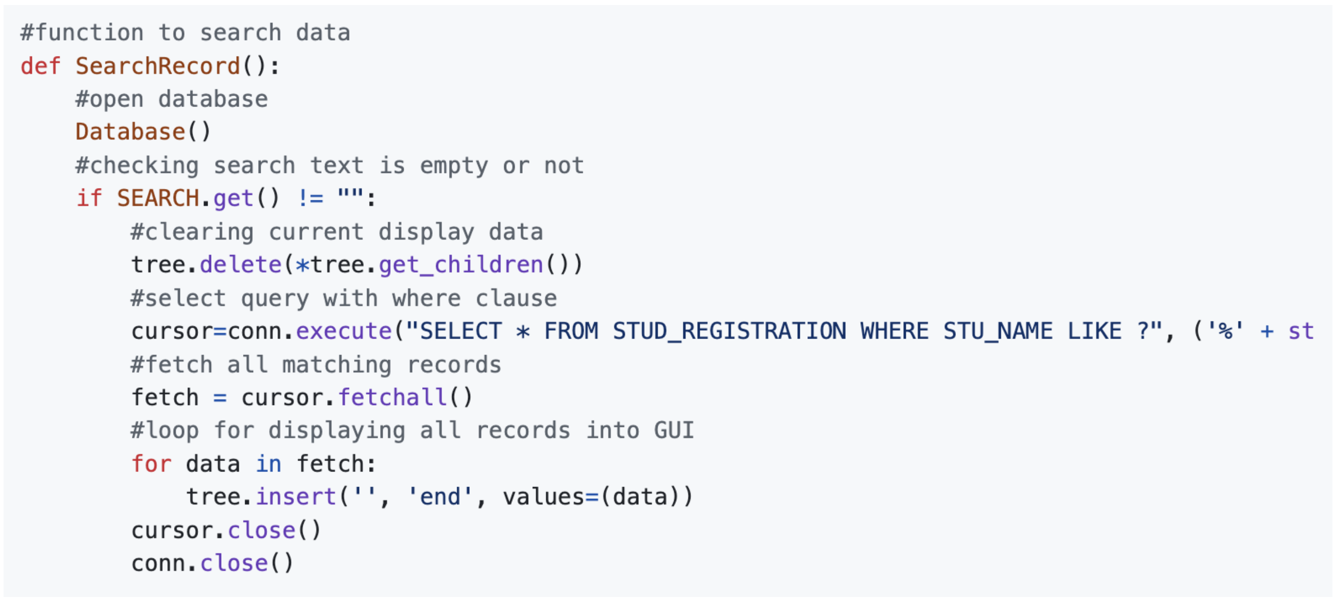
```
def Reset():
    #clear current data from table
    tree.delete(*tree.get_children())
    #refresh table data
    DisplayData()
    #clear search text
    SEARCH.set("")
    name.set("")
    contact.set("")
    email.set("")
    rollno.set("")
    branch.set("")
```

Delete student record: I have defined a function Delete() that will take the selected data and will delete it from database. Here is the complete code to delete selected data from database.



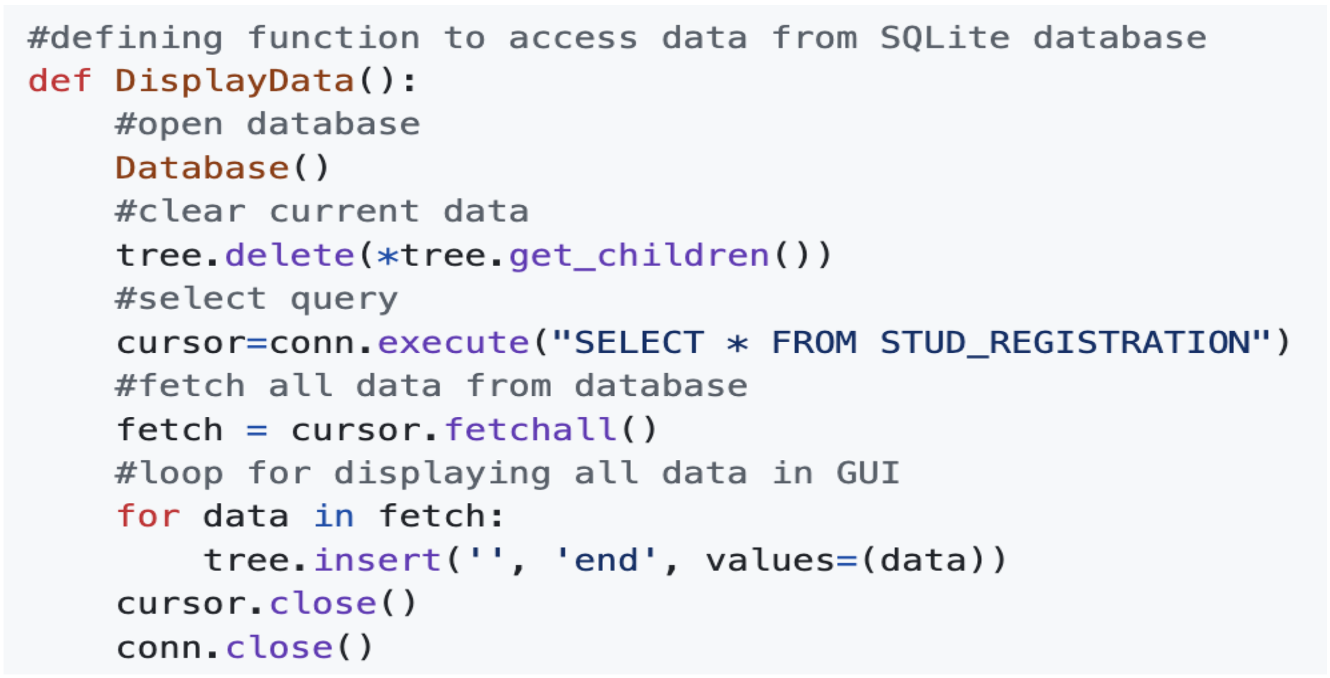
```
def Delete():
    #open database
    Database()
    if not tree.selection():
        tkMessageBox.showwarning("Warning","Select data to delete")
    else:
        result = tkMessageBox.askquestion('Confirm', 'Are you sure you want to delete this rec
                                          icon="warning")
        if result == 'yes':
            curItem = tree.focus()
            contents = (tree.item(curItem))
            selecteditem = contents['values']
            tree.delete(curItem)
            cursor=conn.execute("DELETE FROM STUD_REGISTRATION WHERE STU_ID = %d" % selectedit
            conn.commit()
            cursor.close()
            conn.close()
```

Search student record: I have defined a function SearchRecord() that will take the name of student and perform query to select the records of the given student and at last will populate it into the table format.

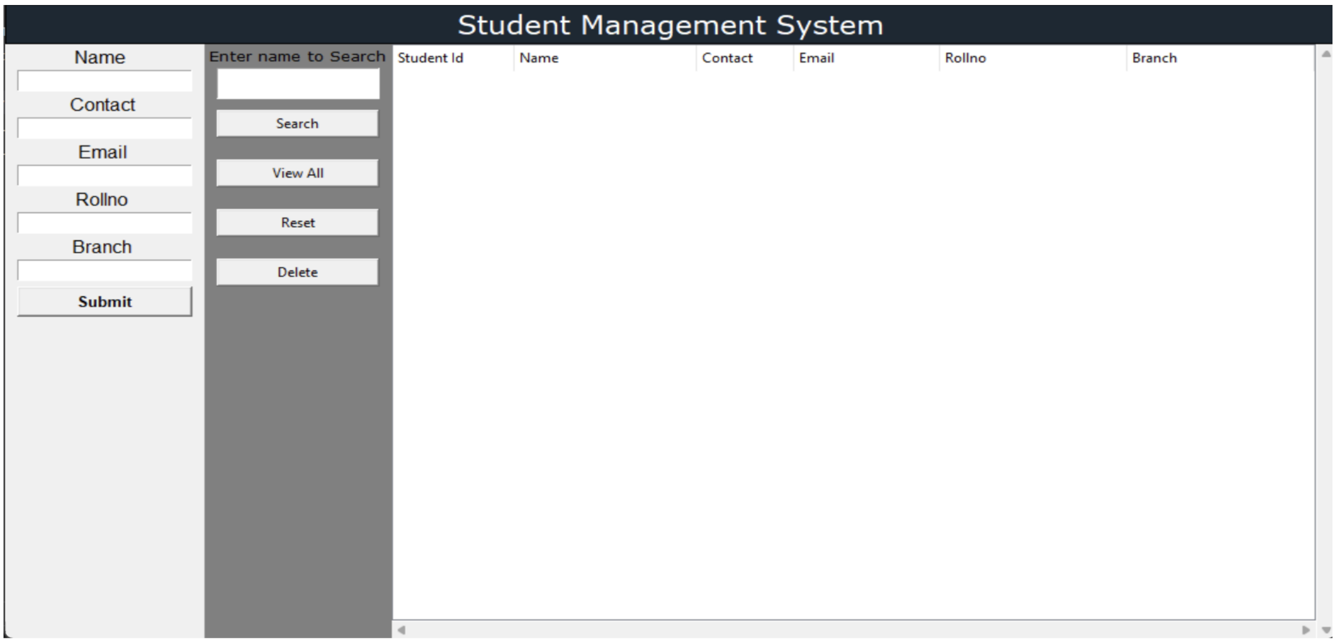


```
#function to search data
def SearchRecord():
    #open database
    Database()
    #checking search text is empty or not
    if SEARCH.get() != "":
        #clearing current display data
        tree.delete(*tree.get_children())
        #select query with where clause
        cursor=conn.execute("SELECT * FROM STUD_REGISTRATION WHERE STU_NAME LIKE ?", ('%' + st
        #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()
```

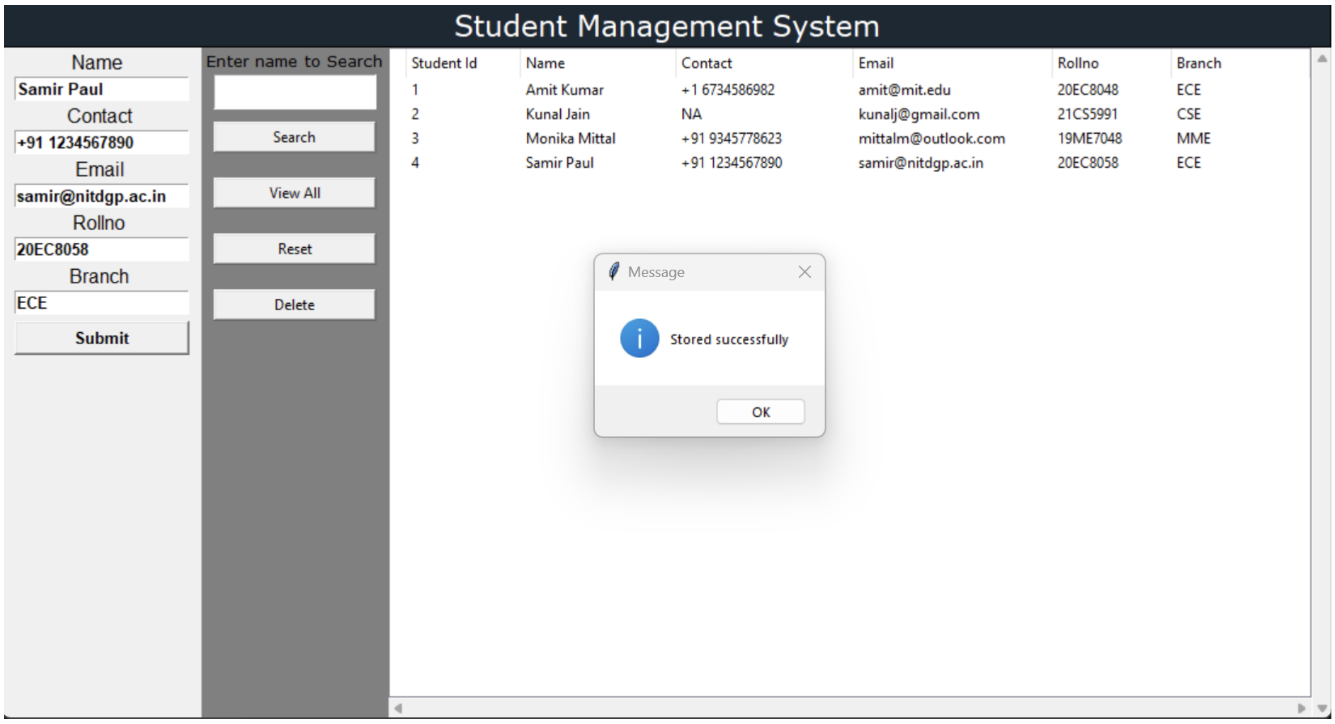
Display student record: I have defined a function DisplayData() that will fetch all the records from the database and display into the GUI in table format. Here is the complete code



```
#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()
```



Insert: Here just fill the Student details and click on Submit button you will get a screen like below

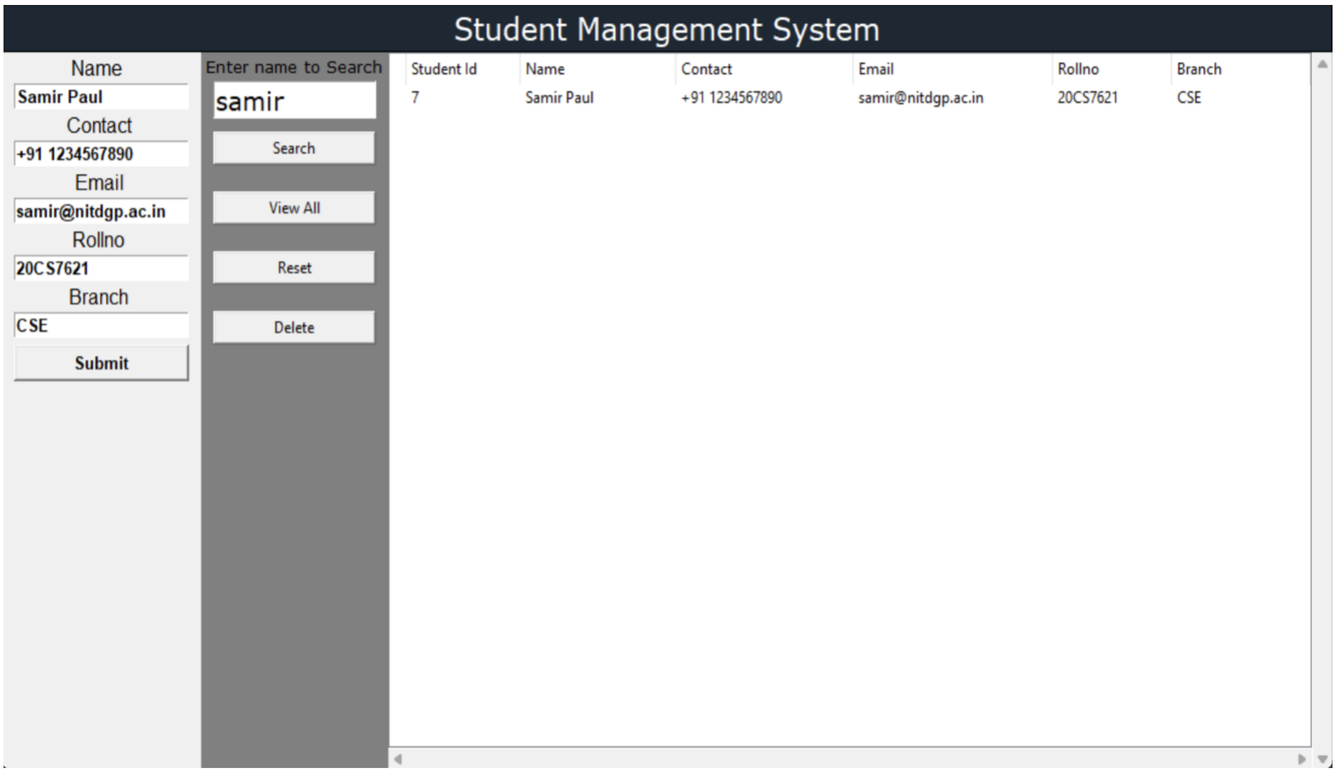


Name: Samir Paul
Contact: +91 1234567890
Email: samir@nitdgp.ac.in
Rollno: 20EC8058
Branch: ECE

Student Id	Name	Contact	Email	Rollno	Branch
1	Amit Kumar	+1 6734586982	amit@mit.edu	20EC8048	ECE
2	Kunal Jain	NA	kunalj@gmail.com	21CS5991	CSE
3	Monika Mittal	+91 9345778623	mittalm@outlook.com	19ME7048	MME
4	Samir Paul	+91 1234567890	samir@nitdgp.ac.in	20EC8058	ECE

Stored successfully

Search: If you want to search record of any student then just fill his/her name in textbox and click on search button. For example here i am going to search student Samir's details

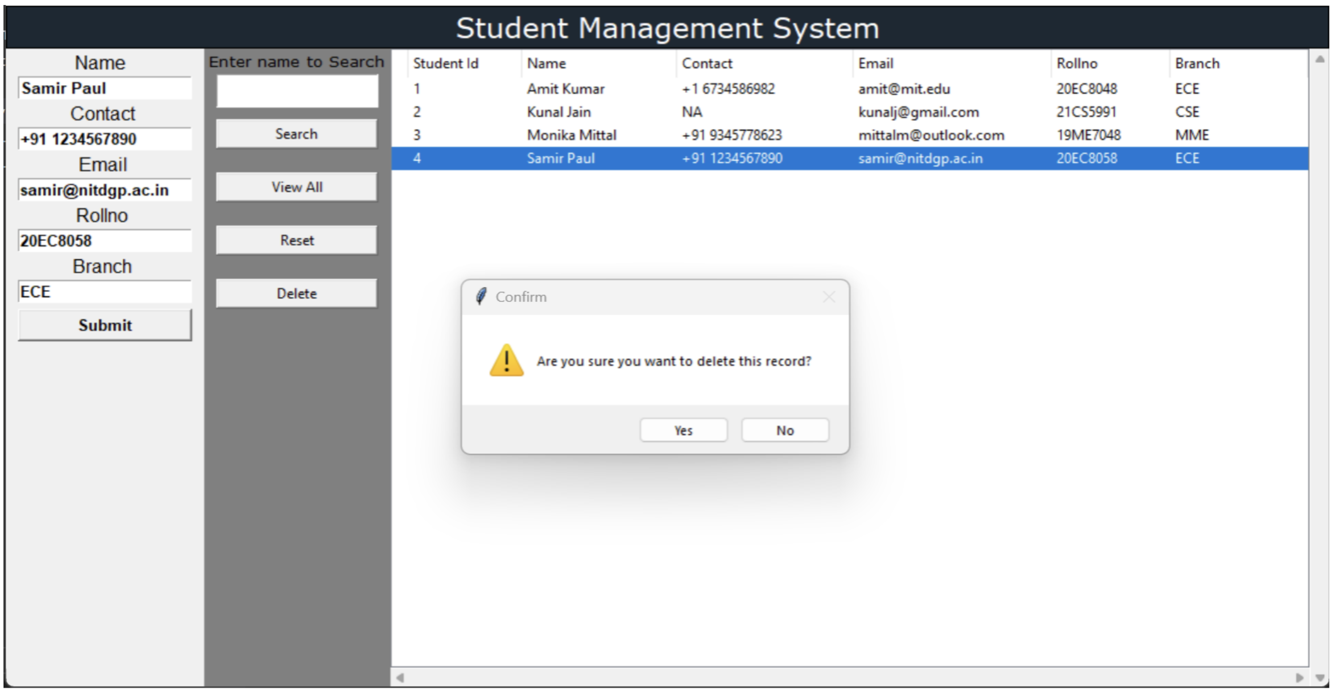

Student Management System

Name	Samir Paul
Contact	+91 1234567890
Email	samir@nitdgp.ac.in
Rollno	20CS7621
Branch	CSE

Enter name to Search: samir

Student Id	Name	Contact	Email	Rollno	Branch
7	Samir Paul	+91 1234567890	samir@nitdgp.ac.in	20CS7621	CSE

Delete:To delete record of any student just click on record to select and the click on delete button.


Student Management System

Name	Samir Paul
Contact	+91 1234567890
Email	samir@nitdgp.ac.in
Rollno	20EC8058
Branch	ECE

Enter name to Search

Student Id	Name	Contact	Email	Rollno	Branch
1	Amit Kumar	+1 6734586982	amit@mit.edu	20EC8048	ECE
2	Kunal Jain	NA	kunalj@gmail.com	21CS5991	CSE
3	Monika Mittal	+91 9345778623	mittalm@outlook.com	19ME7048	MME
4	Samir Paul	+91 1234567890	samir@nitdgp.ac.in	20EC8058	ECE

Confirm

Are you sure you want to delete this record?

DISCUSSION OF RESULTS :

1. Successful Operations:
- If database operations such as adding, updating, or deleting records were successful, it indicates that the integration between the front end and the database is functioning as intended.
- Successful operations imply that data is being accurately captured, stored, and manipulated within the database, ensuring the integrity and consistency of the student management system.

2. Data Consistency and Integrity:
- It's important to verify that the data stored in the database remains consistent and accurate after each operation.
- Ensuring data consistency involves validating that updates or deletions applied to records accurately reflect the intended changes without introducing inconsistencies or data corruption.
- For example, after adding a new student, it's crucial to confirm that all relevant student details are correctly stored in the database and can be retrieved accurately.

3. Error Handling:
- During database operations, encountering errors is common, such as connection issues, syntax errors, or data validation failures.
- Effective error handling mechanisms are essential to gracefully handle these errors, providing informative messages to users and administrators.
- By analyzing error messages, developers can identify the root causes of issues and implement appropriate solutions to prevent recurrence.

4. Performance and Efficiency:
- Evaluating the performance of database operations is crucial to ensure that they meet performance requirements and execute efficiently.
- Monitoring factors like response time, resource utilization, and scalability can help identify performance bottlenecks and optimize database queries for improved efficiency.
- Optimizing database performance enhances the overall responsiveness and scalability of the student management system, ensuring it can handle increased workloads effectively.

5. User Feedback and Validation:
- Gathering feedback from users and stakeholders regarding the usability and effectiveness of database operations is essential for continuous improvement.
- Validating the user experience through testing and soliciting feedback allows developers to identify areas for enhancement and prioritize future development efforts accordingly.
- Incorporating user feedback into the development process ensures that the student management system aligns with user expectations and effectively meets their needs.

In summary, discussing the results obtained from connecting the front end with the database involves evaluating the success of database operations, ensuring data consistency and integrity, addressing error handling mechanisms, optimizing performance, and incorporating user feedback for continuous improvement. This iterative process ensures that the student management system operates reliably and efficiently, providing a seamless user experience for administrators and end users alike.