Faculty of Engineering & Technology			
Ramaiah University of Applied Sciences			
Department	Compute 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.	21ETAI410401	Name of Student	Faisal ali		
Section		**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**		
	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

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:

A.1 Discuss any two database models used in the modern day enterprise computing applications with suitable examples.

Ans : Two Database models are as follows

1. **High-Level or Conceptual Data Model:**
2. **Representational or Implementation Data Model:**

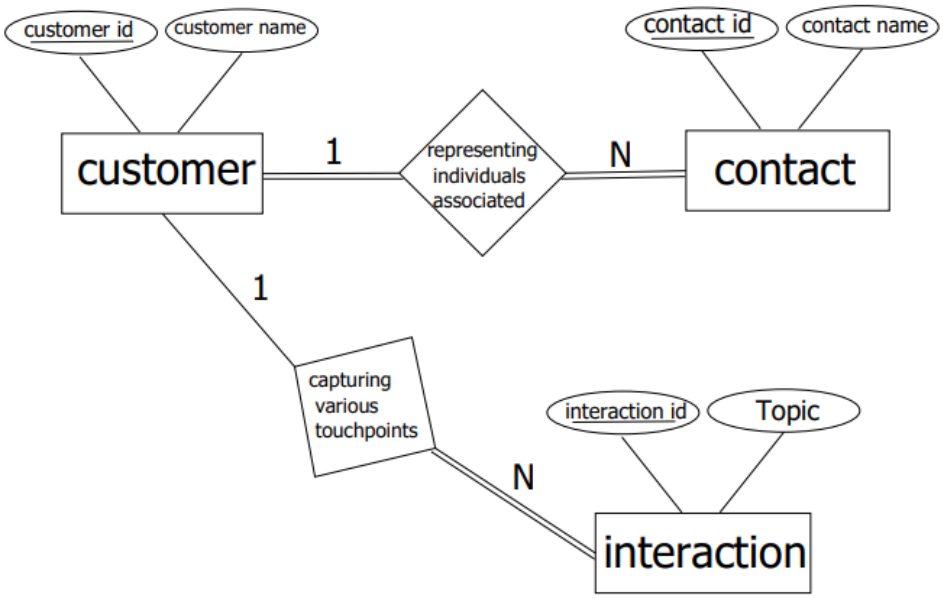
1. High-Level or Conceptual Data Model:

Explanation: A high-level or conceptual data model is an abstract representation of the key entities, relationships, and constraints within a system. It provides a bird's-eye view of the data requirements and focuses on the business concepts without delving into technical details. The primary goal of a conceptual data model is to capture the essential elements of an organization's data architecture and facilitate communication between stakeholders. One commonly used conceptual data modeling technique is the Entity-Relationship Diagram (ERD), where entities represent real-world objects, and relationships illustrate how these entities are interconnected. This level of modeling helps in understanding the overall structure of the data and serves as a foundation for subsequent phases of database design.

Example: **Entity-Relationship Diagram (ERD) for a Customer Relationship Management (CRM) System**

In a CRM system, the conceptual data model through an ERD might include entities such as "Customer," "Contact," and "Interaction." Relationships could be established between "Customer" and "Contact" (representing individuals associated with the customer), as well as between "Customer" and "Interaction" (capturing various touchpoints with the customer). Attributes may include "CustomerID," "ContactName," and "InteractionDate." This high-level model helps in understanding the core entities and their interconnections in managing customer relationships.

Entity-Relationship Diagram (ERD) for a Customer Relationship Management (CRM) System



Significance: The conceptual model guides stakeholders in visualizing how customer-related data is structured and interconnected, aiding in the design of CRM functionalities.

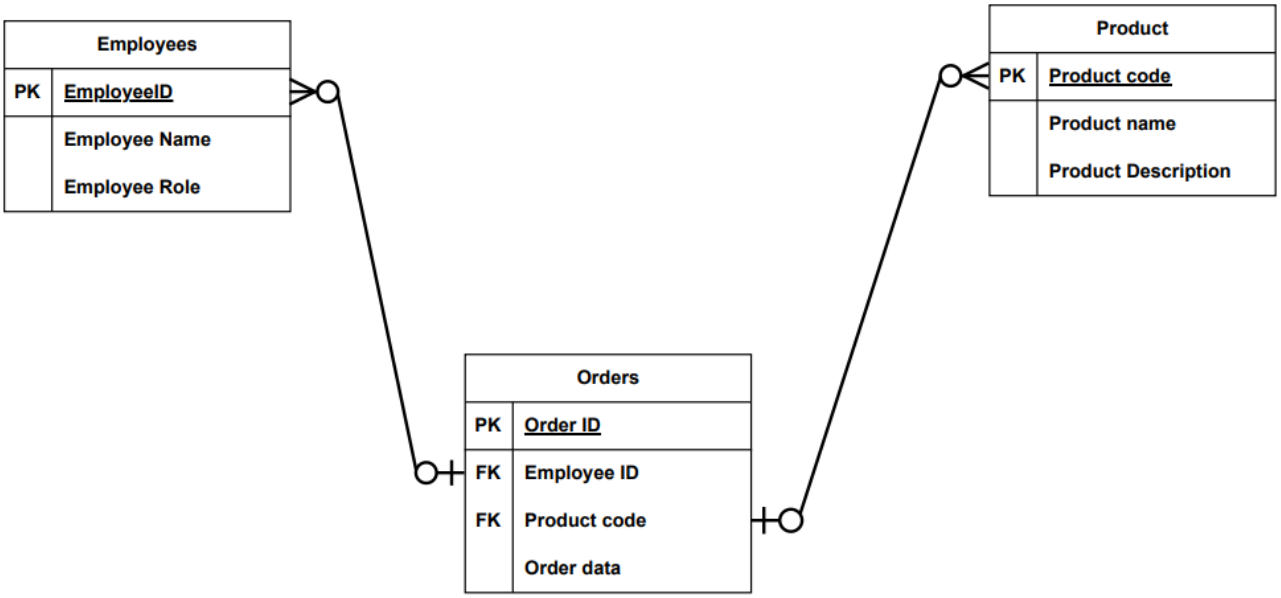
2. Representational or Implementation Data Model:

Explanation: The representational or implementation data model is a more detailed and technical depiction of how the data will be organized and stored in a database system. It translates the concepts outlined in the conceptual data model into a format that can be implemented in a specific database management system (DBMS). One example of a representational data model is the Relational Data Model, which defines tables, attributes, and relationships with a focus on the actual implementation of a relational database. In this phase, considerations such as data types, indexing, and normalization are addressed to optimize database performance. The representational data model serves as a blueprint for database developers and administrators to implement the database structure based on the conceptual model.

Example: Relational Database Model for an Enterprise Resource Planning (ERP) System

For an ERP system, the representational data model could be based on the relational database model. Tables such as "Employees," "Products," and "Orders" may be defined. Columns like "EmployeeID," "ProductCode," and "OrderDate" would capture specific data attributes. Relationships would be established through foreign keys, linking, for instance, the "Employees" table to the "Orders" table. This implementation model provides the specifics on how data is structured, stored, and related within the ERP database.

Relational Database Model for an Enterprise Resource Planning (ERP) System



Significance: The representational model serves as a blueprint for database developers, guiding the actual creation and management of the database to support ERP functionalities.

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

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

B2.3 Design and implementation of GUI

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

Note: Make appropriate assumptions to make the specification complete.

Ans :

B2.1 List of functional requirements

Functionalities that have been taken into consideration are as follows

- **Student Details:**

  a. Student name: maintaining a record of all student names who have enrolled to the university

  b. Email id: maintaining a record of all student id

  c. Phone number: maintaining a record of student's phone number

  d. Gender: getting the information about students' gender

  e. Student id: Assigning the student a student and maintaining it as the primary reference for the identification of students

- **Branch Details:**

  a. Student Branch: maintaining the record of the which branch respective student have joined

- **Fee Payment:**

  a. Fees: maintaining a record how much a student must pay entire year fees and how much he had paid

- **Exam Results:**

a. Exam result: Calculating the aggregate and maintaining the SGPA of each student

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

Creation of data base and table with the help of SQL commands

```sql
create database RUAS_Student_details ;

use RUAS_Student_details ;

create table student_details (
Student_Name varchar(50),
Email_id varchar(50),
Phone_No varchar(50),
Gender varchar(50),
Student_ID_No varchar(50) primary key,
Select_Branch varchar(50),
fees varchar(50),
Exam_result varchar(50)
);

select*from student_details;
```

Result Grid

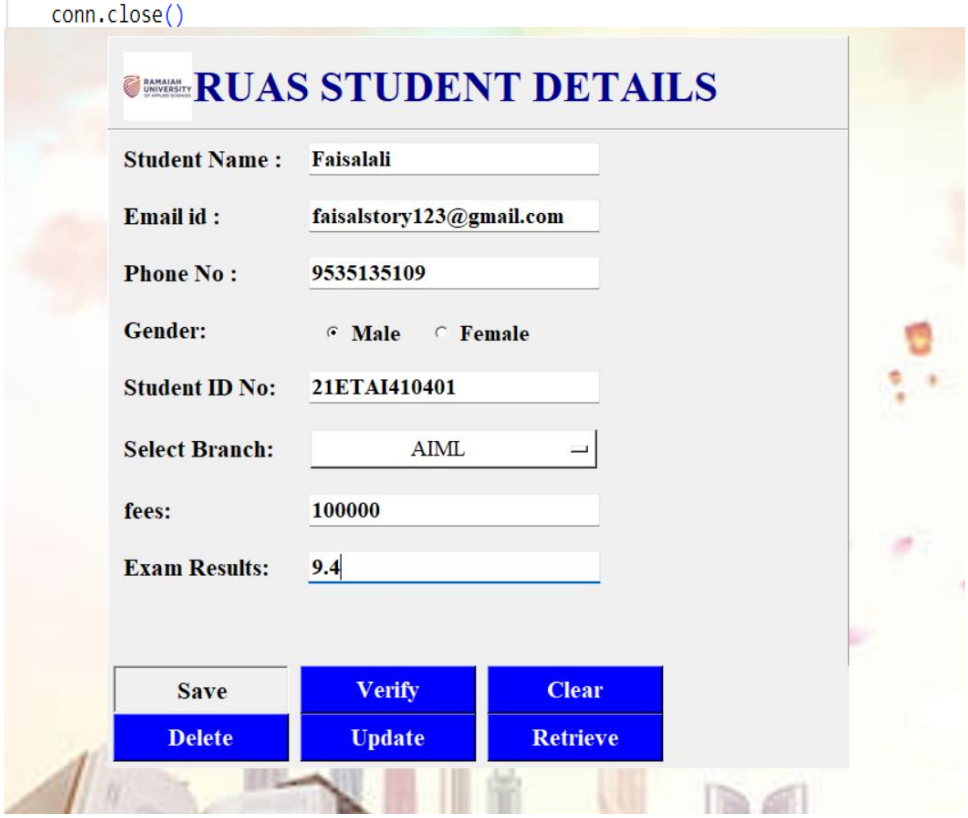
Student_Name	Email_id	Phone_No	Gender	Student_ID_No	Select_Branch	fees	Exam_result
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Insertion of data into the table can be done by connecting the GUI to particular database and the table DML commands have been used the above .

CRUD operations are performed

Insertion of the data:

```
try:
    conn = mysql.connector.connect(host='localhost', username='faisalali', password='742233', database='RUAS_Student_details')
    my_cursor = conn.cursor()

    # Corrected the SQL query by replacing '.' with ','
    my_cursor.execute('INSERT INTO student_details VALUES (%s, %s, %s, %s, %s, %s, %s, %s)', (
        self.name.get(),
        self.emailid.get(),
        self.phonenum.get(),
        self.gend.get(),
        self.studentid.get(),
        self.bran.get(),
        self.feees.get(),
        self.examresuult.get()
    ))

    conn.commit()
    messagebox.showinfo('Success', 'Data has been saved successfully!')
    conn.close()
```



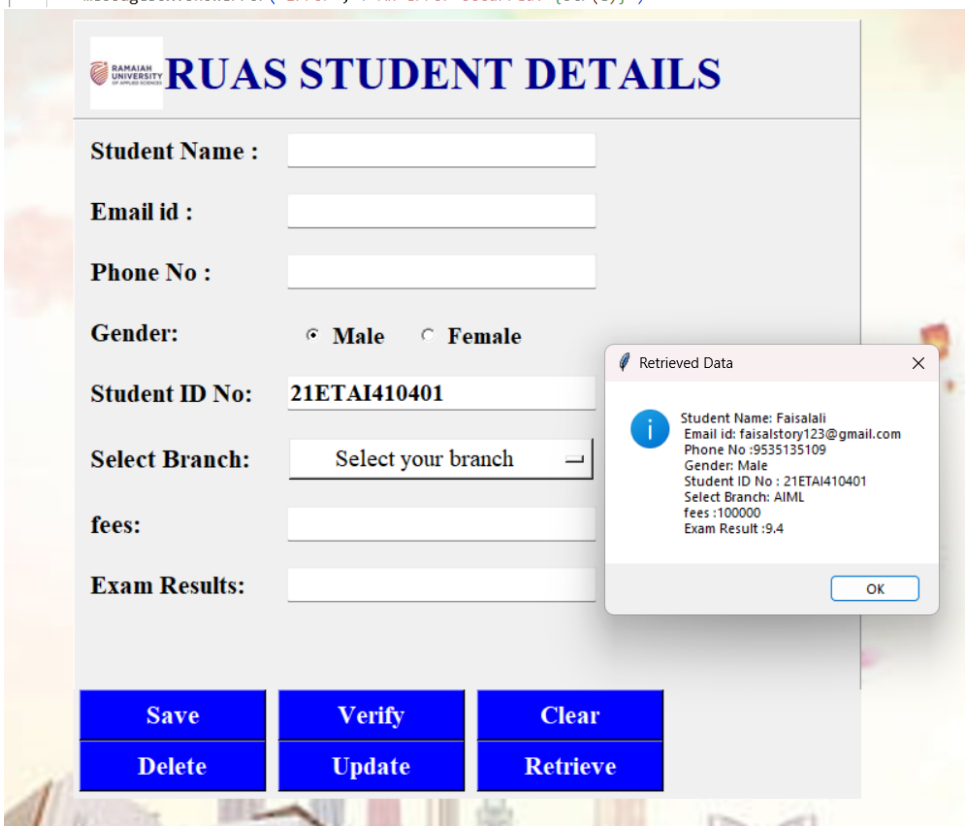
RAMAIAH UNIVERSITY OF APPLIED SCIENCES

```
SELECT * FROM ruas_student_details.student_details;
```

Student_Name	Email_id	Phone_No	Gender	Student_ID_No	Select_Branch	fees	Exam_result
Faisalali	faisalstory123@gmail.com	9535135109	Male	21ETAI410401	AIML	100000	9.4
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

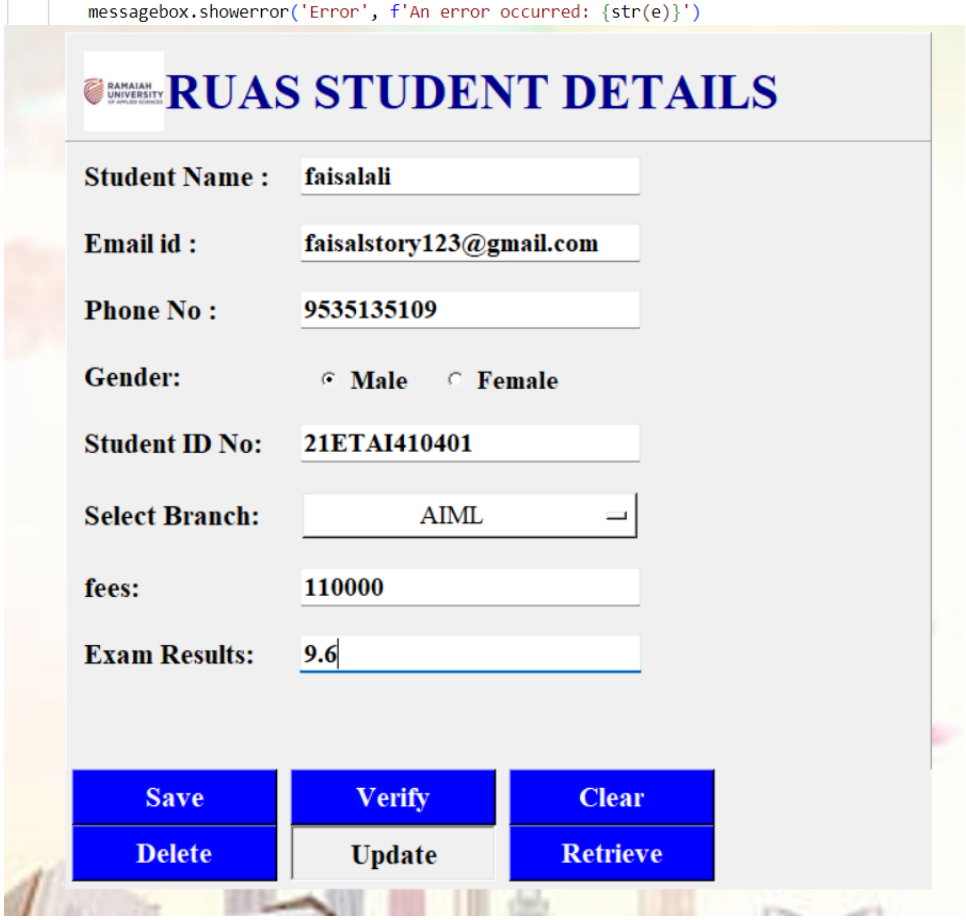
Retrieval of the data: (with the help of primary key)

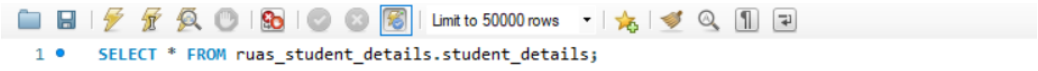
```python
def retrieve_data(self):
    try:
        selected_student_id = self.studentid.get()


        if selected_student_id:
            conn = mysql.connector.connect(host='localhost', username='faisalali', password='742233', database='RUAS_Student_details')
            my_cursor = conn.cursor()

            my_cursor.execute('SELECT * FROM student_details WHERE Student_ID_No = %s', (selected_student_id,))
            data = my_cursor.fetchone()

            conn.close()

            if data:
            # Display the retrieved data in a messagebox
                retrieved_data = f'Student Name: {data[0]}\n Email id: {data[1]}\n Phone No :{data[2]}\n Gender: {data[3]}\n Student ID No : {data[4]}\n
                messagebox.showinfo('Retrieved Data', retrieved_data)
            else:
                messagebox.showinfo('Info', 'No data found for the given Student ID')

        else:
            messagebox.showerror('Error', 'Please enter the Student ID to retrieve', parent=self.root)

    except Exception as e:
        messagebox.showerror('Error', f'An error occurred: {str(e)}')
```



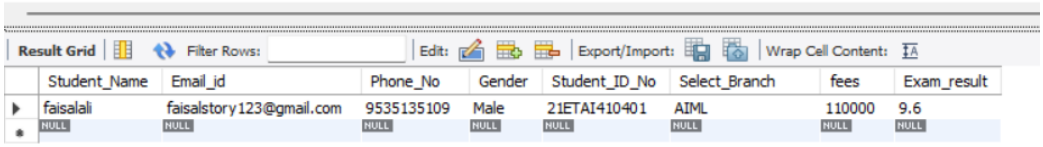
Updating of the data:

```python
def update_data(self):
    # Function to update data in the database
    try:
        conn = mysql.connector.connect(host='localhost', username='faisalali', password='742233', database='RUAS_Student_details')
        my_cursor = conn.cursor()

        # Get the selected student ID for update
        selected_student_id = self.studentid.get()

        # Check if a student ID is selected
        if selected_student_id:
            # Corrected the SQL query by replacing '.' with ','
            my_cursor.execute(
                'UPDATE student_details SET Student_Name=%s, Email_id=%s, Phone_No=%s, Gender=%s, Select_Branch=%s, fees=%s, Exam_result=%s
                (self.name.get(), self.emailid.get(), self.phonenum.get(), self.gend.get(), self.bran.get(),
                 self.feees.get(), self.examresuult.get(), selected_student_id))
            conn.commit()
            messagebox.showinfo('Success', 'Data has been updated successfully!')
            conn.close()
        else:
            messagebox.showerror('Error', 'Please enter the Student ID to update', parent=self.root)

    except Exception as e:
        messagebox.showerror('Error', f'An error occurred: {str(e)}')
```






```
1 •  SELECT * FROM ruas_student_details.student_details;
```



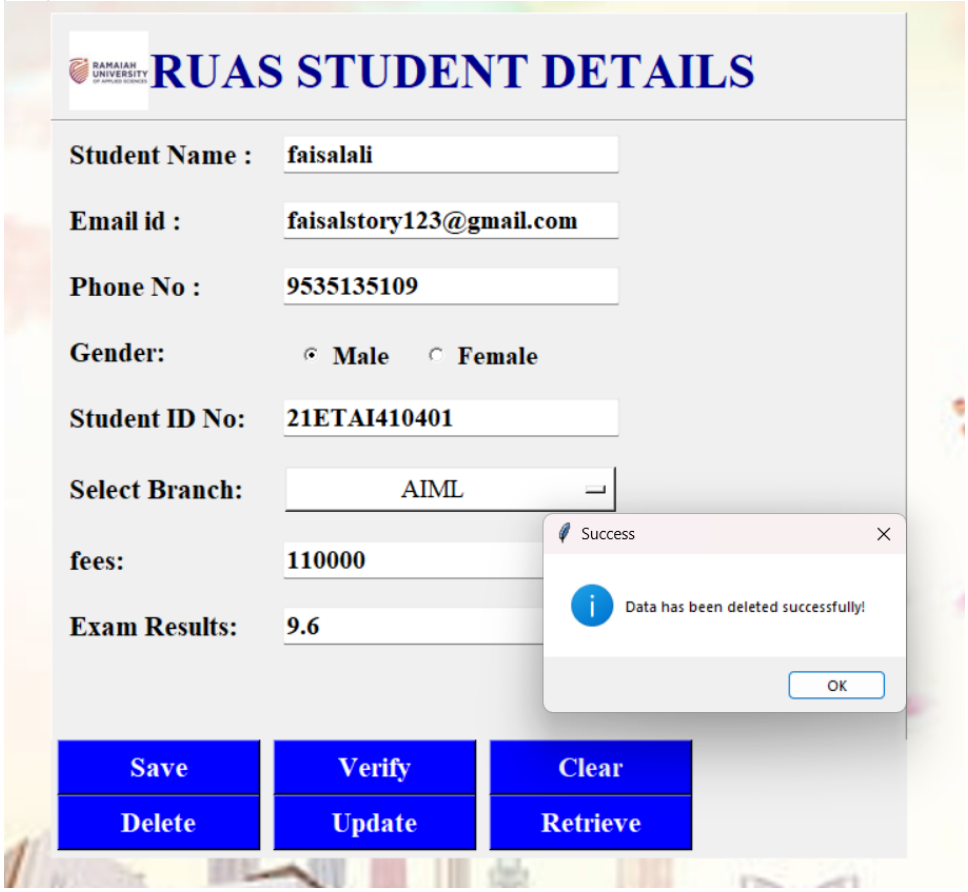
Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

Student_Name	Email_id	Phone_No	Gender	Student_ID_No	Select_Branch	fees	Exam_result
faisalali	faisalstory123@gmail.com	9535135109	Male	21ETAI410401	AIML	110000	9.6
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fees and Exam_results have been updated

Deletion of the data:

```
def delete_data(self):
    # Function to delete data from the database
    try:
        conn = mysql.connector.connect(host='localhost', username='faisalali', password='742233', database='RUAS_Student_details')
        my_cursor = conn.cursor()

        # Get the selected student ID for deletion
        selected_student_id = self.studentid.get()

        # Check if a student ID is selected
        if selected_student_id:
            # Corrected the SQL query by replacing '.' with ','
            my_cursor.execute('DELETE FROM student_details WHERE Student_ID_No = %s', (selected_student_id,))
            conn.commit()
            messagebox.showinfo('Success', 'Data has been deleted successfully!')
            conn.close()
            self.clear_data()  # Clear the entries after deletion
        else:
            messagebox.showerror('Error', 'Please enter the Student ID to delete', parent=self.root)

    except Exception as e:
        messagebox.showerror('Error', f'An error occurred: {str(e)}')
```



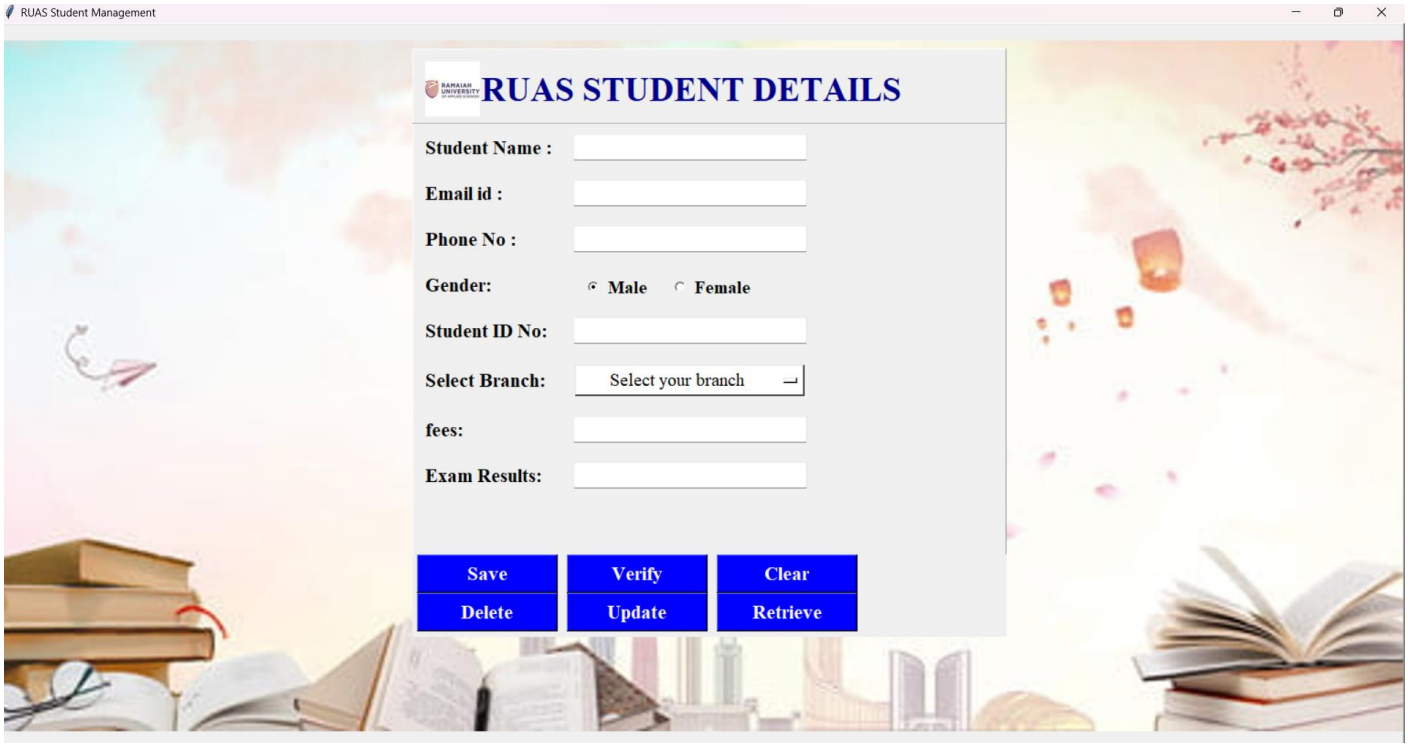

```
1 •  SELECT * FROM ruas_student_details.student_details;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

Student_Name	Email_id	Phone_No	Gender	Student_ID_No	Select_Branch	fees	Exam_result
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

B2.3 Design and implementation of GUI

GUI frontend



Code for the frontend:

```
from tkinter import *
from tkinter import ttk
from PIL import Image, ImageTk
from tkinter import messagebox
import re
import mysql.connector

class Register():
    def __init__(self, root):
        self.root = root
        self.root.title("RUAS Student Management")
        self.root.geometry("1600x790+0+0")

        # creation of text variable for the input entry
        self.name = StringVar()
        self.emailid = StringVar()
        self.phonenum = StringVar()
        self.gend = StringVar()
        self.studentid = StringVar()
        self.bran = StringVar()
        self.feees = StringVar()
        self.examresuult = StringVar()
        self.checkvar = IntVar()

        # Adding a background image
        self.bg = ImageTk.PhotoImage(file='bg8.jpg')
```

```
        bg_lbl = Label(self.root, image=self.bg, bd=2, relief=RAISED)
        bg_lbl.place(x=0, y=0, relwidth=1, relheight=1)

        # Logo image
        logo_img = Image.open('logo.png')
        # resizing the photo
        logo_img = logo_img.resize((60, 60), Image.LANCZOS)
        # setting the photo
        self.photo_logo = ImageTk.PhotoImage(logo_img)

        # Title frame
        title_frame = Frame(self.root, bd=1, relief=RIDGE)
        title_frame.place(x=450, y=28, width=650, height=82)

        title_lbl = Label(title_frame, image=self.photo_logo, compound=LEFT,
text='RUAS STUDENT DETAILS', font=('times new roman', 28, 'bold'), fg='darkblue')
        title_lbl.place(x=10, y=10)

        # Information frame
        main_frame = Frame(self.root, bd=1, relief=RIDGE)
        main_frame.place(x=450, y=110, width=650, height=560)

         #Student name
        user_name = Label(main_frame,text ='Student Name :',font=('times new
roman',16,'bold'))
        user_name.grid(row=0,column=0,padx=10,pady=10,sticky=W)
        #Student name entry
        user_entry =ttk.Entry(main_frame,textvariable=self.name,font=('times new
roman',15,'bold'),width =25)
        user_entry.grid(row=0,column=1,padx=10,pady=10,sticky=W)
        #bind/callback  and validation with register
        validate_name = self.root.register(self.checkname)
        user_entry.config(validate='key',validatecommand=(validate_name,'%P'))


        #Email id
        Email_name = Label(main_frame,text ='Email id :',font=('times new
roman',16,'bold'))
        Email_name.grid(row=1,column=0,padx=10,pady=10,sticky=W)
        #Student name entry
        Email_entry =ttk.Entry(main_frame,textvariable=self.emailid,font=('times new
roman',15,'bold'),width =25)
        Email_entry.grid(row=1,column=1,padx=10,pady=10,sticky=W)
        #call back and validating
        validate_email = self.root.register(self.checkemail)
        Email_entry.config(validate='key',validatecommand=(validate_email,'%P'))

        #contact
        contact = Label(main_frame,text ='Phone No :',font=('times new
roman',16,'bold'))
        contact.grid(row=2,column=0,padx=10,pady=10,sticky=W)
        #contact entry
```

```
        contact_entry =ttk.Entry(main_frame,textvariable=self.phonenum ,font=('times
new roman',15,'bold'),width =25)
        contact_entry.grid(row=2,column=1,padx=10,pady=10,sticky=W)
        #call back and validating
        validate_phone = self.root.register(self.phonenumber)
        contact_entry.config(validate='key',validatecommand=(validate_phone,'%P'))


        #Student Gender (Radio buttons)
        Gender= Label(main_frame,text ='Gender:',font=('times new roman',16,'bold'))
        Gender.grid(row=3,column=0,padx=10,pady=10,sticky=W)
        #student gender entry
        # Gender_entry =ttk.Entry(main_frame,font=('times new roman',15,'bold'),width
=25)
        # Gender_entry.grid(row=3,column=1,padx=10,pady=10,sticky=W)

        Gender_entry = Frame(main_frame) #creation of the gender entry frame
        Gender_entry.place(x=176,y=160,width=280,height=40)


radio_male=Radiobutton(Gender_entry,variable=self.gend,value='Male',text='Male',font=(
'times new roman',15,'bold'))
        radio_male.grid(row=0,column=0,padx=10,pady=0,sticky=W)


radio_female=Radiobutton(Gender_entry,variable=self.gend,value='Female',text='Female',
font=('times new roman',15,'bold'))
        radio_female.grid(row=0,column=1,padx=10,pady=0,sticky=W)
        self.gend.set("Male")



        #Student id number
        StudentID = Label(main_frame,text ='Student ID No:',font=('times new
roman',16,'bold'))
        StudentID.grid(row=4,column=0,padx=10,pady=10,sticky=W)
        #student id number entry
        StudentID_entry =ttk.Entry(main_frame,textvariable =self.studentid,
font=('times new roman',15,'bold'),width =25)
        StudentID_entry.grid(row=4,column=1,padx=10,pady=10,sticky=W)
        #checkin and validation for student id
        validate_studentid = self.root.register(self.student_id)

StudentID_entry.config(validate='key',validatecommand=(validate_studentid,'%P'))

        #Select branch
        branch = Label(main_frame,text ='Select Branch:',font=('times new
roman',16,'bold'))
        branch.grid(row=5,column=0,padx=10,pady=10,sticky=W)
        #select branch entry
        # branch_entry =ttk.Entry(main_frame,font=('times new roman',15,'bold'),width
=25)
```

```
        # branch_entry.grid(row=5,column=1,padx=10,pady=10,sticky=W)

        list1 = ['ASE','AIML','Computer science','ISE','MC','Civil','Mechanical']
        droplist =OptionMenu(main_frame,self.bran,*list1)
        droplist.config(width=21,font=('times new roman',15),bg ='white')
        self.bran.set('Select your branch')
        # droplist.place(x=240,y=420)
        droplist.grid(row=5,column=1,padx=10,pady=10,sticky=W)


        #Fee
        fee = Label(main_frame,text ='fees:',font=('times new roman',16,'bold'))
        fee.grid(row=6,column=0,padx=10,pady=10,sticky=W)
        #Fee entry
        fee_entry =ttk.Entry(main_frame,textvariable=self.feees,font=('times new
roman',15,'bold'),width =25)
        fee_entry.grid(row=6,column=1,padx=10,pady=10,sticky=W)
        #checkin and validation for fee
        validate_fee = self.root.register(self.feenumber)
        fee_entry.config(validate='key',validatecommand=(validate_fee,'%P'))

        #Exam results
        exam = Label(main_frame,text ='Exam Results:',font=('times new
roman',16,'bold'))
        exam.grid(row=7,column=0,padx=10,pady=10,sticky=W)
        #Exam results entry
        exam_entry =ttk.Entry(main_frame,textvariable=self.examresuult,font=('times
new roman',15,'bold'),width =25)
        exam_entry.grid(row=7,column=1,padx=10,pady=10,sticky=W)
        #checin and validation of exam results
        validate_results = self.root.register(self.exam_result)
        exam_entry.config(validate='key',validatecommand=(validate_results,'%P'))


        # Creating a frame for the delete and update buttons
        btn_frame = Frame(self.root)
        btn_frame.place(x=450, y=580, width=650, height=90)

        # Save button
        save_data = Button(btn_frame, text='Save', command=self.validation,
font=('times new roman', 16, 'bold'), width=12, cursor='hand2', bg='blue', fg='white')
        save_data.grid(row=0, column=0, padx=5, sticky=W)

        # Verify button
        verify_data = Button(btn_frame, command=self.verify_data, text='Verify',
font=('times new roman', 16, 'bold'), width=12, cursor='hand2', bg='blue', fg='white')
        verify_data.grid(row=0, column=1, padx=5, sticky=W)

        # Clear button
        clear_data = Button(btn_frame, command=self.clear_data, text='Clear',
font=('times new roman', 16, 'bold'), width=12, cursor='hand2', bg='blue', fg='white')
        clear_data.grid(row=0, column=2, padx=5, sticky=W)
```

```
        # Delete button
        delete_data = Button(btn_frame, command=self.delete_data, text='Delete',
font=('times new roman', 16, 'bold'), width=12, cursor='hand2', bg='blue', fg='white')
        delete_data.grid(row=1, column=0, padx=5, sticky=W)

        # Update button
        update_data = Button(btn_frame, command=self.update_data, text='Update',
font=('times new roman', 16, 'bold'), width=12, cursor='hand2', bg='blue', fg='white')
        update_data.grid(row=1, column=1, padx=5, sticky=W)

        #retrieve data button
        retrieve_data = Button(btn_frame, command=self.retrieve_data, text='Retrieve',
font=('times new roman', 16, 'bold'), width=12, cursor='hand2', bg='blue', fg='white')
        retrieve_data.grid(row=1, column=2, padx=5, sticky=W)

    # ... (existing code)

    # Call back function (binding)
    def checkname(self, name):
        if name.isalnum():  # a to z and from 0 to 9
            return True
        if name == '':
            return True
        else:
            messagebox.showerror('Invalid', 'Not Allowed ' + name[-1])

    # ... (existing code)

    # Check for the email id
    def checkemail(self, email):
        if len(email) > 3:
            if re.match("^([a-zA-Z0-9_\-\.]+)@([a-zA-Z0-9_\-\.]+)\.([a-zA-Z]{2,5})$",
email):
                return True
            else:
                messagebox.showwarning('Alert', 'invalid email enter valid user email
(example : faisalali11@gmail.com)')
                return False
        else:
            messagebox.showinfo('Invalid', 'Email length is too small')

    # Check for phone number
    def phonenumber(self, phone):
        if phone.isdigit():
            return True
        if len(str(phone)) == 0:
            return True
        else:
            messagebox.showerror('Invalid', "Invalid Entry")
            return False
```

```
    # Check for the student id
    def student_id(self, studentid):
        if studentid.isalnum():  # a to z and from 0 to 9
            return True
        if studentid == '':
            return True
        else:
            messagebox.showerror('Invalid', 'Not Allowed ' + studentid[-1])

    # Check for the fees
    def feenumber(self, fee):
        if fee.isdigit():
            return True
        if len(str(fee)) == 0:
            return True
        else:
            messagebox.showerror('Invalid', "Invalid Entry")
            return False

    # Check for the exam results
    def exam_result(self, result):
        try:
            float(result)
            return True
        except ValueError:
            messagebox.showerror('Invalid', "Invalid Entry. Please enter a valid
number.")
            return False

    # =============== main validation data entry part ==============
    def validation(self):
        if self.name.get() == '':
            messagebox.showerror('Error', 'Please enter the name ', parent=self.root)

        elif self.emailid.get() == '':
            messagebox.showerror('Error', 'Please enter your email ',
parent=self.root)

        elif self.phonenum.get() == '' or len(self.phonenum.get()) != 10:
            messagebox.showerror('Error', 'Please enter your valid Phone number',
parent=self.root)

        elif self.gend.get() == '':
            messagebox.showerror('Error', 'Please select the gender ',
parent=self.root)

        elif self.studentid.get() == '':
            messagebox.showerror('Error', 'Please enter the student id ',
parent=self.root)

        elif self.bran.get() == '' or self.bran.get() == 'Select your branch':
            messagebox.showerror('Error', 'please select your branch ',
```

```
parent=self.root)

        elif self.feees.get() == '':
            messagebox.showerror('Error', 'Please enter the fees ', parent=self.root)

        elif self.examresuult.get() == '':
            messagebox.showerror('Error', 'Please enter the result ',
parent=self.root)

        elif self.emailid.get() != None:
            if not self.checkemail(self.emailid.get()):
                return  # If email is invalid, return without proceeding to MySQL

        # ==================== connecting to MySQL ================
        try:
            conn = mysql.connector.connect(host='localhost', username='faisalali',
password='742233', database='RUAS_Student_details')
            my_cursor = conn.cursor()

            # Corrected the SQL query by replacing '.' with ','
            my_cursor.execute('INSERT INTO student_details VALUES (%s, %s, %s, %s, %s,
%s, %s, %s)', (
                self.name.get(),
                self.emailid.get(),
                self.phonenum.get(),
                self.gend.get(),
                self.studentid.get(),
                self.bran.get(),
                self.feees.get(),
                self.examresuult.get()
            ))

            conn.commit()
            messagebox.showinfo('Success', 'Data has been saved successfully!')
            conn.close()

        except Exception as e:
            messagebox.showerror('Error', f'An error occurred: {str(e)}')

    def verify_data(self):
        data = f'Student Name: {self.name.get()}\n Email id: {self.emailid.get()}\n
Phone No :{self.phonenum.get()}\n Gender: {self.gend.get()}\n Student ID No :
{self.studentid.get()}\n Select Branch: {self.bran.get()}\n fees :{self.feees.get()}\n
Exam Result :{self.examresuult.get()}'
        messagebox.showinfo('Details', data)

    def clear_data(self):
        self.name.set('')
        self.emailid.set('')
        self.phonenum.set('')
        self.gend.set('Male')
        self.studentid.set('')
```

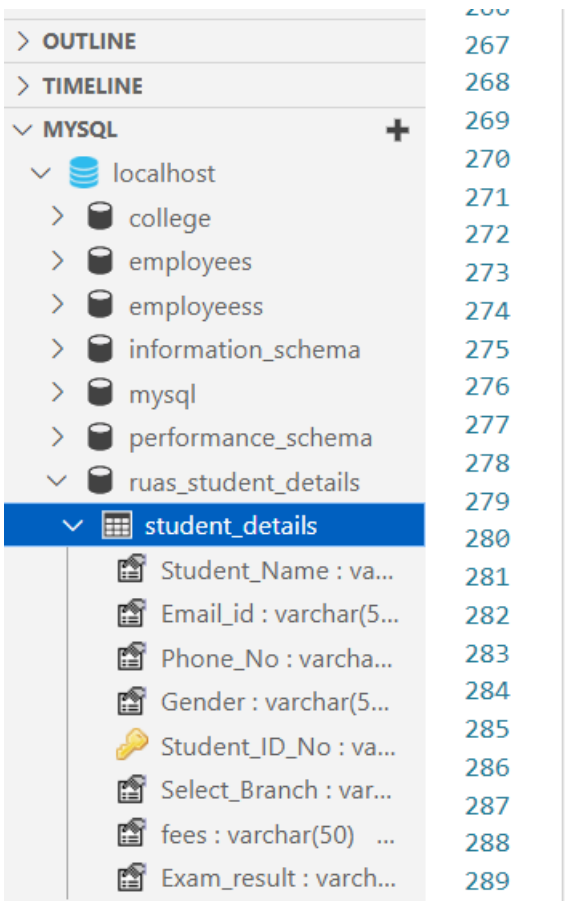
```
        self.bran.set('Select your branch')
        self.feees.set('')
        self.examresuult.set('')

    def delete_data(self):
        # Function to delete data from the database
        try:
            conn = mysql.connector.connect(host='localhost', username='faisalali',
password='742233', database='RUAS_Student_details')
            my_cursor = conn.cursor()

            # Get the selected student ID for deletion
            selected_student_id = self.studentid.get()

            # Check if a student ID is selected
            if selected_student_id:
                # Corrected the SQL query by replacing '.' with ','
                my_cursor.execute('DELETE FROM student_details WHERE Student_ID_No =
%s', (selected_student_id,))
                conn.commit()
                messagebox.showinfo('Success', 'Data has been deleted successfully!')
                conn.close()
                self.clear_data()  # Clear the entries after deletion
            else:
                messagebox.showerror('Error', 'Please enter the Student ID to delete',
parent=self.root)

        except Exception as e:
            messagebox.showerror('Error', f'An error occurred: {str(e)}')

    def update_data(self):
        # Function to update data in the database
        try:
            conn = mysql.connector.connect(host='localhost', username='faisalali',
password='742233', database='RUAS_Student_details')
            my_cursor = conn.cursor()

            # Get the selected student ID for update
            selected_student_id = self.studentid.get()

            # Check if a student ID is selected
            if selected_student_id:
                # Corrected the SQL query by replacing '.' with ','
                my_cursor.execute(
                    'UPDATE student_details SET Student_Name=%s, Email_id=%s,
Phone_No=%s, Gender=%s, Select_Branch=%s, fees=%s, Exam_result=%s WHERE Student_ID_No
= %s',
                    (self.name.get(), self.emailid.get(), self.phonenum.get(),
self.gend.get(), self.bran.get(),
                     self.feees.get(), self.examresuult.get(), selected_student_id))
                conn.commit()
                messagebox.showinfo('Success', 'Data has been updated successfully!')
```

```
                conn.close()
            else:
                messagebox.showerror('Error', 'Please enter the Student ID to update',
parent=self.root)

        except Exception as e:
            messagebox.showerror('Error', f'An error occurred: {str(e)}')

    #retrieve button

    def retrieve_data(self):
        try:
            selected_student_id = self.studentid.get()


            if selected_student_id:
                conn = mysql.connector.connect(host='localhost', username='faisalali',
password='742233', database='RUAS_Student_details')
                my_cursor = conn.cursor()

                my_cursor.execute('SELECT * FROM student_details WHERE Student_ID_No =
%s', (selected_student_id,))
                data = my_cursor.fetchone()

                conn.close()

                if data:
                # Display the retrieved data in a messagebox
                    retrieved_data = f'Student Name: {data[0]}\n Email id: {data[1]}\n
Phone No :{data[2]}\n Gender: {data[3]}\n Student ID No : {data[4]}\n Select Branch:
{data[5]}\n fees :{data[6]}\n Exam Result :{data[7]}'
                    messagebox.showinfo('Retrieved Data', retrieved_data)
                else:
                    messagebox.showinfo('Info', 'No data found for the given Student
ID')

            else:
                messagebox.showerror('Error', 'Please enter the Student ID to
retrieve', parent=self.root)

        except Exception as e:
            messagebox.showerror('Error', f'An error occurred: {str(e)}')


if __name__ == "__main__":
    root = Tk()
    app = Register(root)
    root.mainloop()
```

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

```python
try:
    conn = mysql.connector.connect(host='localhost', username='faisalali', password='742233', database='RUAS_Student_details')
    my_cursor = conn.cursor()

    # Corrected the SQL query by replacing '.' with ','
    my_cursor.execute('INSERT INTO student_details VALUES (%s, %s, %s, %s, %s, %s, %s, %s)', (
        self.name.get(),
        self.emailid.get(),
        self.phonenum.get(),
        self.gend.get(),
        self.studentid.get(),
        self.bran.get(),
        self.feees.get(),
        self.examresuult.get()
    ))

    conn.commit()
    messagebox.showinfo('Success', 'Data has been saved successfully!')
    conn.close()

except Exception as e:
    messagebox.showerror('Error', f'An error occurred: {str(e)}')
```

In the same for all CRUD operations the GUI connected with the database



The connection is made through vscode to the MySQL workbench

(frontend to the database and specified table)

Discussion on the results

a. **Save button:** With the help of the save button we can insert the student details with all the constraints applied on each entry.

b. **Verify button:** With the help of verify button we can verify the data once before saving it.

c. **Clear button:** With the help of the clear button, we can clear all data at once.

d. **Delete button:** with reference to the primary key (Student ID No) we can delete any tuple we want with the help of the delete button.

e. **Update button:** with reference to the primary key, we can update the student details easily.

f. **Retrieve button:** with reference to primary key, we can retrieve any of the student details

Conclusion: With the help of all the buttons and constraints on every entry it has become easy to maintain the RUAS Student details

Github link of the project:

https://github.com/faisalali1234567/Student-details-management-GUI