Atma Ram Sanatan Dharma College, DU

SESSION: 2023-24

Python Project: Student Management System;

app.py

```python
members = {
    'name': 'Pawan Kumar' #member1
    'rollno': 37100
    'name': 'Kehsav Dahiya' #member2
    'rollno': 37088
    'name': 'Harsh Kumar' #member3
    'rollno': 37084
    'course' = 'BSc Maths Hons' #courseName
    'section' = 'A' #section
}
```

Submitted to **Manvendra Yadav ;**

STUDENT MANAGEMENT SYSTEM

In the Student Management system Project in python, we will see a project that manages all the information of the students It includes managing data such as name, email-id, contact number, date of birth, which stream they are in, etc.

In this tutorial of the Python project, we will build a GUI-based STUDENT MANAGEMENT SYSTEM Project using the Tkinter, SQLite3, tkcalendar, datetime, messagebox, and datetime and Ttk modules of the Tkinter library. It is an intermediate-level project, where you will learn how to use databases, and modules and make some great GUIs in Python and apply them in real life.

Project Overview

Project Name:	**Students Management Project in Python**
Abstract:	**This is a GUI-based program in python that includes basically make use of the Tkinter and Sqlite database for the execution.**
Language/Technologies used:	**Python, Tkinter**
IDE:	**Pycharm(Recommended), VScode**
Database:	**SQLite3**
Python version:	**3.12.1**
Type/Category:	**1st Sem Project using Python**

Features of Student Management System:-

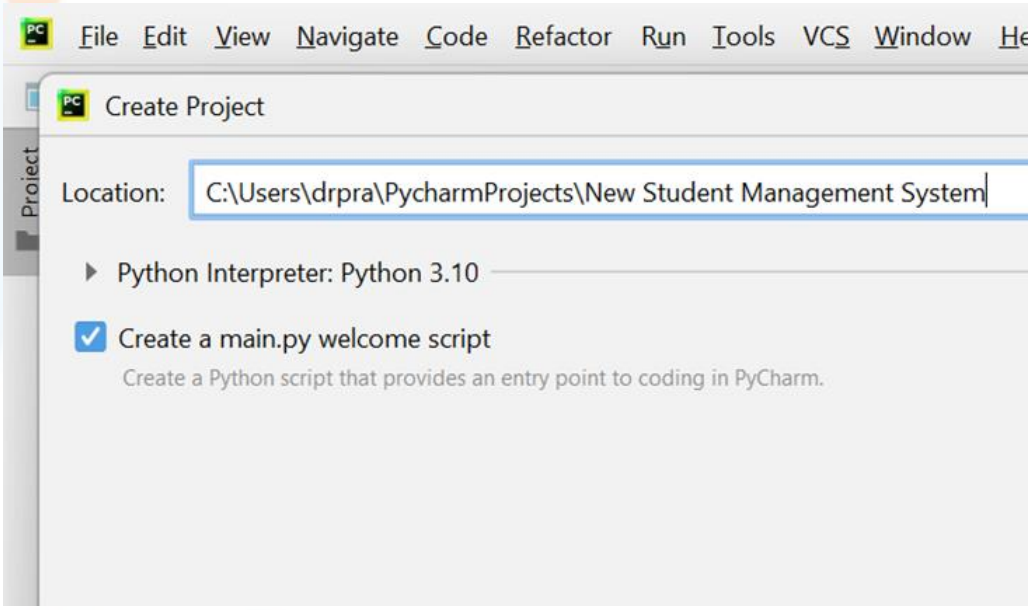
The basic task to be performed on this Project are:

1. **Add all the details**
2. **Update the details**
3. **Reset the details**
4. **Delete the details**
5. **Delete the entire database of the student's record**

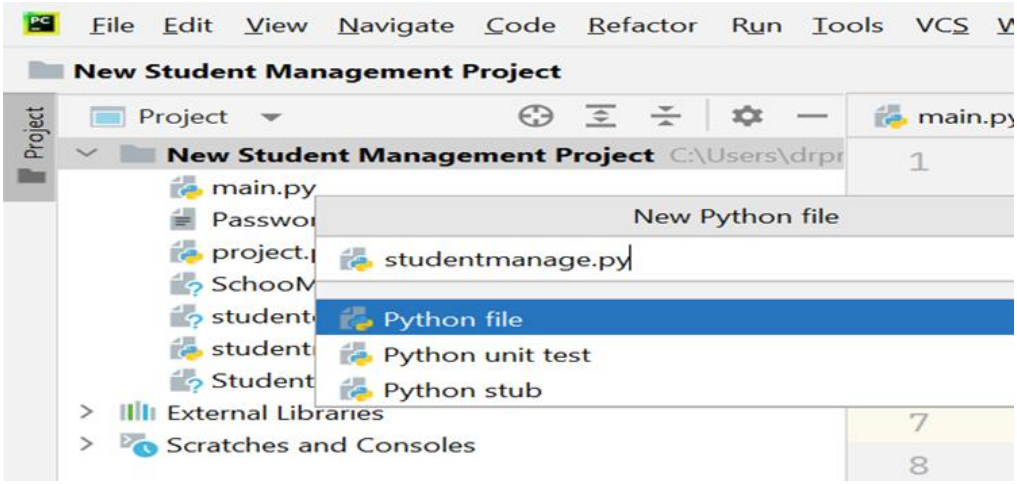
Use of Pycharm IDE for Project

Steps for program execution:

1. **First Install [Pycharm Community Edition 2023.3] (community edition is to be installed)**
2. **Create New Project by clicking on File and selecting New Project, writing the project name, and entering "Create".**



3. **Right-click on the project name you have created and Create a New Python File as "studentmanage.py"**



4. **Write the code in the file and execute the Python Program for the student management system by Clicking the Run tab.**

***Note*: Before Importing the tkcalendar in the program install it by using the command " pip install tkcalendar" and rest libraries that are not imported, by using the terminal/command prompt.**

You can also install the modules by going to "File"->" Settings"-> "Project: New Student Management"->" Python Interpreter"->click on the"+" sign and write the name of the module want to install.

Now let us understand the source code of the Student Management System in Python using Tkinter in detail.

Code flow: Student Management System in python with source code

Importing the Libraries

app.py

```
import datetime
from tkinter import *
import tkinter.messagebox as mb
from tkinter import ttk
from tkcalendar import DateEntry  # pip install tkcalendar
import sqlite3
```

Explanation:

These modules are used for the following purposes:

1. **Tkinter – To create the GUI.**
2. **SQLite3 – To connect the program to the database and store information in it.**
3. **TkCalender – To get the user to enter a date.**
4. **Datetime.date – To convert the date from the tree to a Datetime.date instance so that it can be set in.**
5. **Tkinter.messagebox – To show a display box, displaying some information or an error.**
6. **Tkinter.ttk – To create the tree where all the information will be displayed.**

Create the font variable and perform database connectivity

app.py

```
# Creating the variables
headlabelfont = ("Calibri", 15, 'bold')
labelfont = ('Calibri', 14)
entryfont = ('Calibri', 14)


# Connecting to the Database where all information will be stored
connector = sqlite3.connect('Studentmanagement.db')
cursor = connector.cursor()
connector.execute(
"CREATE TABLE IF NOT EXISTS STUDENT_MANAGEMENT (STUDENT_ID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, NAME TEXT, EMAIL TEXT, PHONE_NO TEXT, GENDER TEXT, DOB TEXT, STREAM TEXT) "
)
```

Explanation:

Creating the universal font variables:

We have created a universal font for all the variables in the entire program. Here we have mentioned the Font, style, and size of the variables in the code.

Connecting to the database:

The connect() function accepts connection credentials and returns an object of type SQLITE for database connection. The mysql.cursor() is used to communicate with the SQLite. connector.execute() will create the table in the database if it doesn't exist.

Creation of GUI Window for Student Management system

app.py

```
# Initializing the GUI window
main = Tk()
main.title('Student Management System')
main.geometry('1000x800')
main.resizable(0, 0)

# Creating the background and foreground color variables
lf_bg = 'SteelBlue' # bg color for the left_frame

# Creating the StringVar or IntVar variables
name_strvar = StringVar()
email_strvar = StringVar()
contact_strvar = StringVar()
gender_strvar = StringVar()
stream_strvar = StringVar()

# Placing the components in the main window
Label(main, text="STUDENT MANAGEMENT SYSTEM", font='Arial', bg='SkyBlue').pack(side=TOP, fill=X)
left_frame = Frame(main, bg=lf_bg)
left_frame.place(x=0, y=30, height=1000, width=400)
right_frame = Frame(main, bg="gray")
right_frame.place(x=400, y=30, height=1000, width=600)


# Placing components in the left frame
Label(left_frame, text="Name", font=labelfont, bg=lf_bg).place(x=30, y=50)
Label(left_frame, text="Contact Number", font=labelfont, bg=lf_bg).place(x=30,y=100)
Label(left_frame, text="Email Address", font=labelfont, bg=lf_bg).place(x=30,y=150)
Label(left_frame, text="Gender", font=labelfont, bg=lf_bg).place(x=30, y=200)
Label(left_frame, text="Date of Birth (DOB)", font=labelfont, bg=lf_bg).place(x=30, y=250)
Label(left_frame, text="Stream", font=labelfont, bg=lf_bg).place(x=30, y=300)
Entry(left_frame, width=20, textvariable=name_strvar, font=entryfont).place(x=170, y=50)
Entry(left_frame, width=19, textvariable=contact_strvar, font=entryfont).place(x=170, y=100)
Entry(left_frame, width=19, textvariable=email_strvar, font=entryfont).place(x=170,y=150)
Entry(left_frame, width=19, textvariable=stream_strvar, font=entryfont).place(x=170, y=300)
OptionMenu(left_frame, gender_strvar, 'Male', "Female").place(x=170, y=200, width=70)
dob = DateEntry(left_frame, font=("Arial", 12), width=15)
dob.place(x=180, y=250)
Button(left_frame, text='Submit and Add Record', font=labelfont, command=add_record, width=18).place(x=80, y=380)

#Place the buttons in the left frame

Button(left_frame, text='Delete Record', font=labelfont, command=remove_record, width=15).place(x=30, y=450)
Button(left_frame, text='View Record', font=labelfont, command=view_record, width=15).place(x=200, y=450)
Button(left_frame, text='Clear Fields', font=labelfont, command=reset_fields, width=15).place(x=30, y=520)
Button(left_frame, text='Remove database', font=labelfont, command=reset_form, width=15).place(x=200, y=520)
# Placing components in the right frame

Label(right_frame, text='Students Records', font='Arial', bg='DarkBlue', fg='LightCyan').pack(side=TOP, fill=X)
tree = ttk.Treeview(right_frame, height=100, selectmode=BROWSE,
                    columns=('Stud ID', "Name", "Email Addr", "Contact No", "Gender", "Date of Birth", "Stream"))
X_scroller = Scrollbar(tree, orient=HORIZONTAL, command=tree.xview)
Y_scroller = Scrollbar(tree, orient=VERTICAL, command=tree.yview)
X_scroller.pack(side=BOTTOM, fill=X)
Y_scroller.pack(side=RIGHT, fill=Y)
tree.config(yscrollcommand=Y_scroller.set, xscrollcommand=X_scroller.set)

tree.heading('Stud ID', text='ID', anchor=CENTER)
tree.heading('Name', text='Name', anchor=CENTER)
tree.heading('Email Addr', text='Email ID', anchor=CENTER)
tree.heading('Contact No', text='Phone No', anchor=CENTER)
tree.heading('Gender', text='Gender', anchor=CENTER)
tree.heading('Date of Birth', text='DOB', anchor=CENTER)
tree.heading('Stream', text='Stream', anchor=CENTER)
tree.column('#0', width=0, stretch=NO)
tree.column('#1', width=40, stretch=NO)
tree.column('#2', width=120, stretch=NO)
tree.column('#3', width=180, stretch=NO)
tree.column('#4', width=60, stretch=NO)
tree.column('#5', width=60, stretch=NO)
tree.column('#6', width=70, stretch=NO)
tree.column('#7', width=120, stretch=NO)
tree.place(y=30, relwidth=1, relheight=0.9, relx=0)
display_records()


main.update()
main.mainloop()
```

Explanation:

Initialzing the GUI Window

We have initialized the GUI Window by creating the object of TK() as the main. Set the title of the window, The geometry is the size of the window.

Creating the color variables: Creating the background and foreground color variables.

Creating the StringVar and IntVar variables: Stores the name, email, gender, contact, and stream you entered in the form.

Place the components in the main window

We are creating the main frame where we are labeling the window with the title "Student Management System".We are also creating 2 frames into that i.e left_frame and right frame specifying the color, width, and height of the frame.

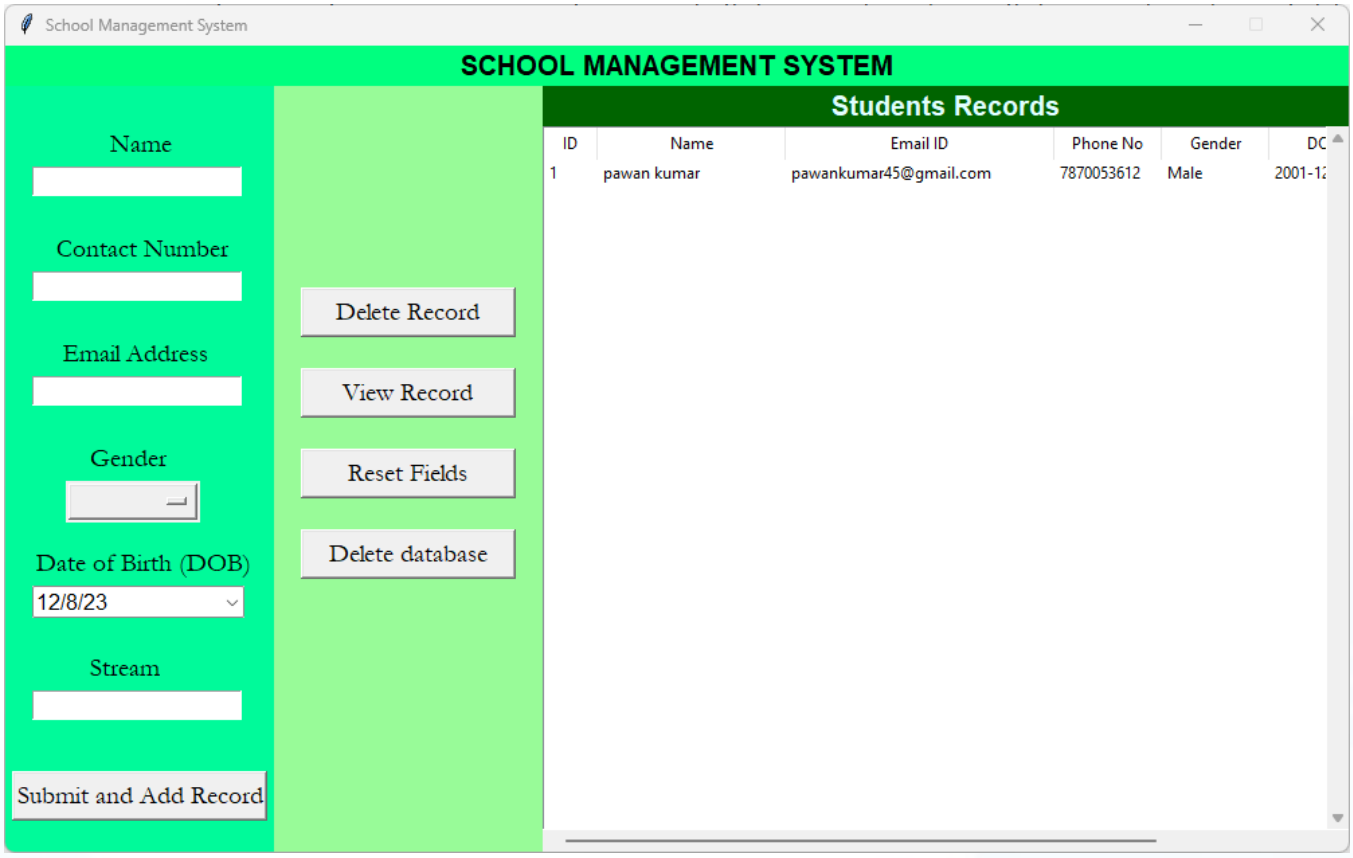
Placing components in the left frame

Here we are placing the components like name, contact number, email address, gender, date of birth, and Stream The label function is used for labeling the components and provides formatting. Entry function is used for entering the text.OptionMenu provides the dropdown list. The Button label is used for the data entry we provided.

Placing components in the Right frame

The right frame displays the data that we have entered, deleted, reset, or modified. We have provided the scroll_bar to scroll the details horizontally and vertically.

Output:-



Creating a Reset Function

app.py

```
# Create the reset function
def reset_fields():
    global name_strvar, email_strvar, contact_strvar, gender_strvar, dob, stream_strvar
    for i in ['name_strvar', 'email_strvar', 'contact_strvar', 'gender_strvar', 'stream_strvar']:
        exec(f"{i}.set('')")
    dob.set_date(datetime.datetime.now().date())
```

Explanation:

Reset fields: The reset fields mention the variables whose value is to be reset. Those are defined as global variables in the program.

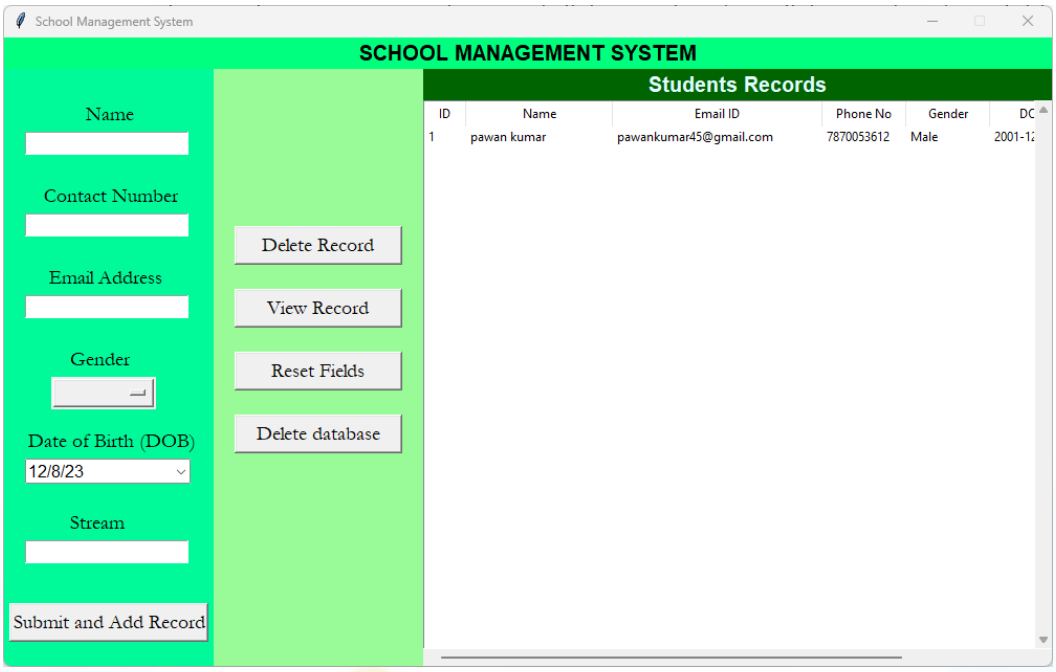
Displays the records in the database

app.py

```
#function to display records
def display_records():
    tree.delete(*tree.get_children())
    c = connector.execute('SELECT * FROM STUDENT_MANAGEMENT')
    data = c.fetchall()
    for records in data:
        tree.insert('', END, values=records)
```

Explanation:

The display_records: In This function we create the object curr which obtains the credentials after executing the database query. The c.fetchall() fetches all the data and stores it in the data variable. The for loop function executes for each record in the data and inserted into the fields of the table



Add and Submit the records to the database

app.py

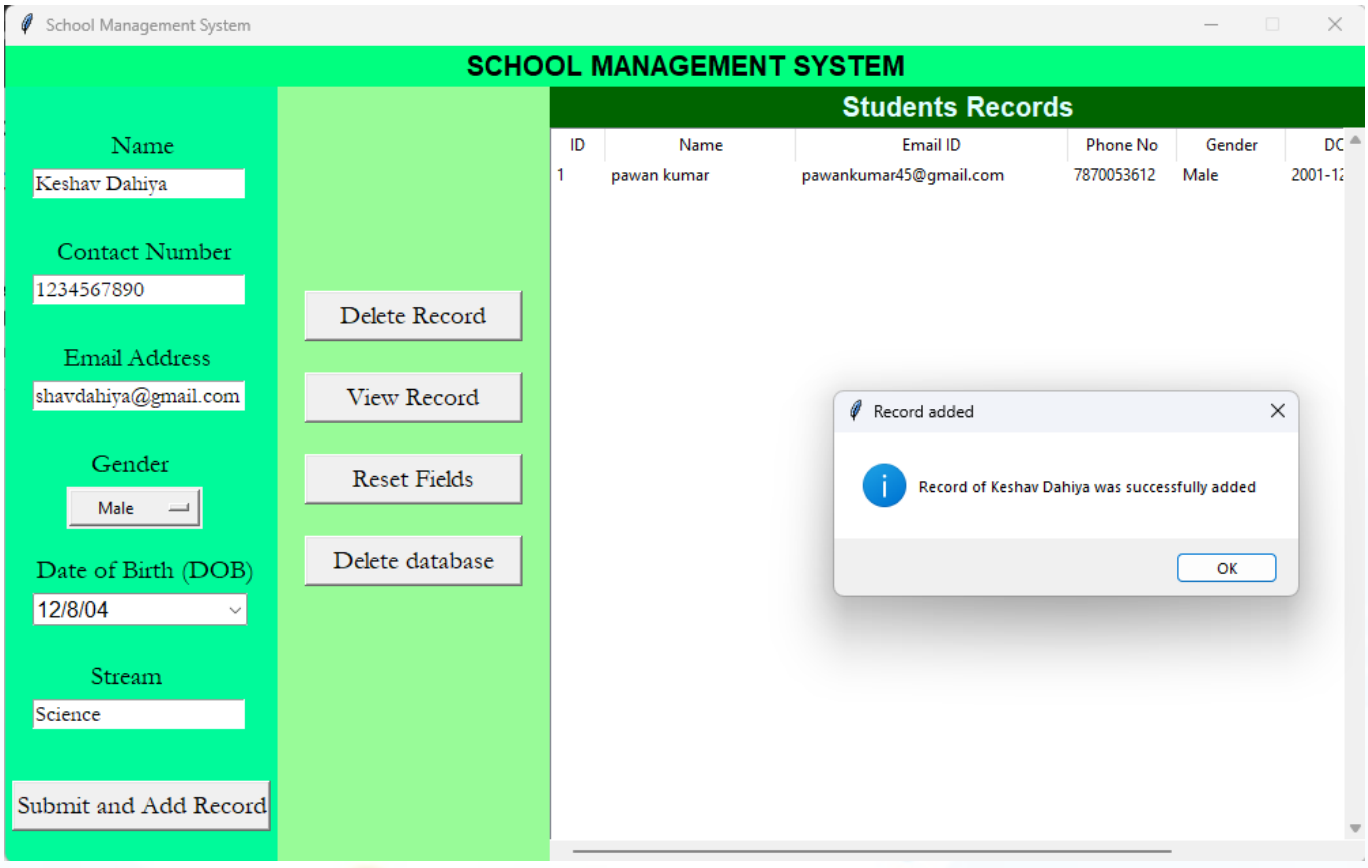
```
#Function to add record
def add_record():
    global name_strvar, email_strvar, contact_strvar, gender_strvar, dob, stream_strvar
    name = name_strvar.get()
    email = email_strvar.get()
    contact = contact_strvar.get()
    gender = gender_strvar.get()
    DOB = dob.get_date()
    stream = stream_strvar.get()
    if not name or not email or not contact or not gender or not DOB or not stream:
        mb.showerror('Error!', "Please enter all the details!")
    else:
        try:
            connector.execute(
            'INSERT INTO Student_MANAGEMENT (NAME, EMAIL, PHONE_NO, GENDER, DOB, STREAM) VALUES (?,?,?,?,?,?) ', (name, email, contact, gender, DOB, stream)
            )
            connector.commit()
            mb.showinfo('Record inserted', f"Record of {name} is added")
            reset_fields()
            display_records()
        except:
            mb.showerror('Wrong type', 'The contact number should be 10 digits ')
```

Explanation:

Add_records:

The global variables initialized here, using the get function takes the values from the form and store them in the local variables provided. In this process, if any of the values i.e name, email, contact, gender, dob, or stream is not filled it will show the error as 'Error!', "Please fill all the missing fields!!") else if all the text fields are properly fielded then the record is added into the database and displays the message" 'Record added', f"Record of {name} is added". The contact no field is restricted to 10 digits.

Output:-



Delete a record of the student from the database

app.py

```
#Function to remove record
def remove_record():
    if not tree.selection():
        mb.showerror('Error!', 'Please select an item from the database')
    else:
        current_item = tree.focus()
        values = tree.item(current_item)
        selection = values["values"]
        tree.delete(current_item)
        connector.execute('DELETE FROM STUDENT_MANAGEMENT WHERE STUDENT_ID=%d' % selection[0])
        connector.commit()
        mb.showinfo('Done', 'The record is deleted successfully.')
        display_records()
```

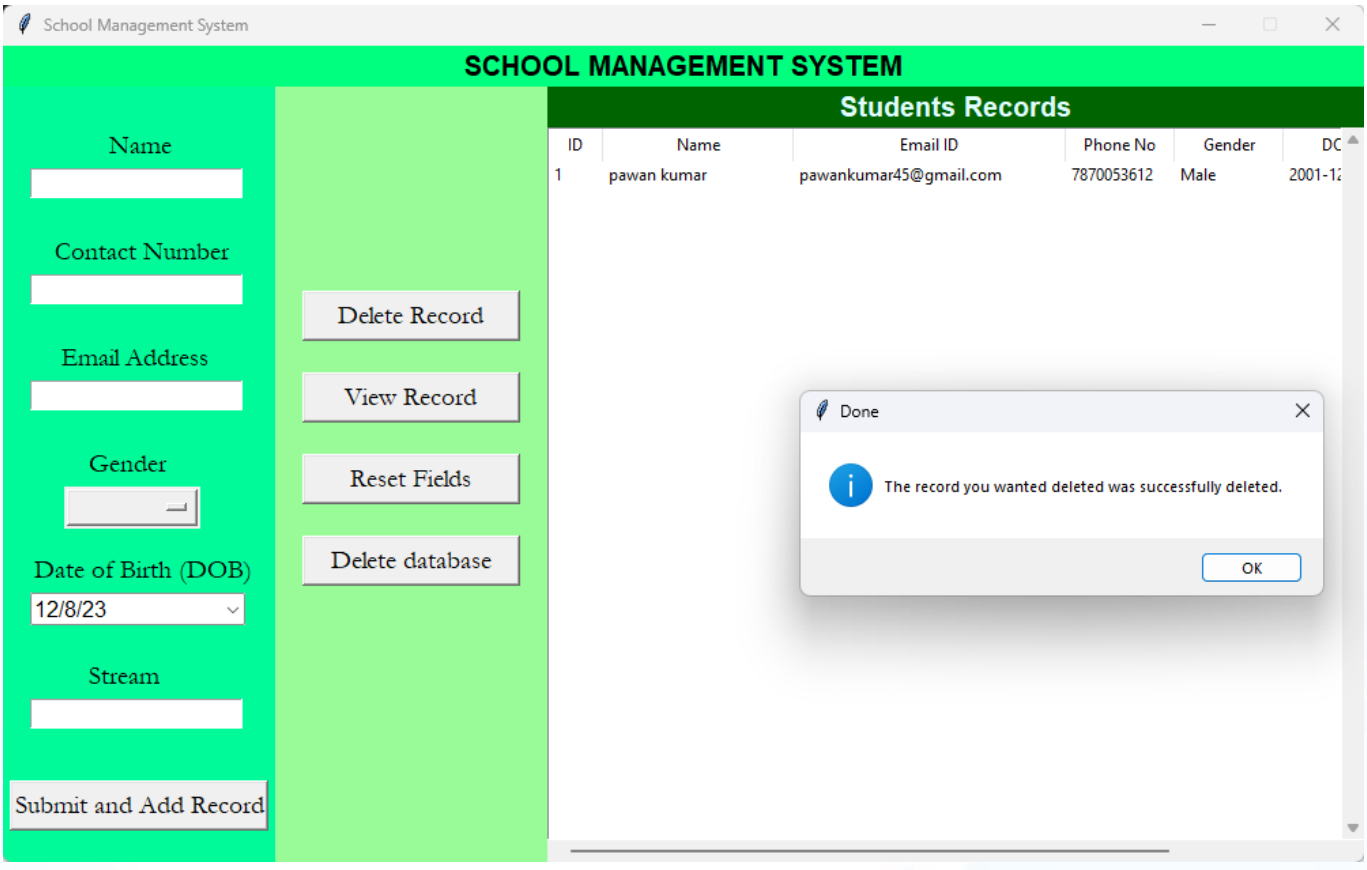
Explanation:

The remove_record():

This function checks to see if the record is selected to be removed else the tree.focus() function selects the current_item that is in focus.Puts all the values in the selection variable where the entire row of the focused current_item gets deleted by executing the SQL Query.

Once the row is deleted it shows the message "The record is successfully deleted" and displays the remaining records.

Output:-



Delete the Database of the Student Management system

app.py

```
def reset_form():
    global tree
    tree.delete(*tree.get_children())

   reset_fields()
```

Explanation:

Reset_form(): The tree variable is initialized as global. The tree.delete() delete all the records of the database and resets the field().

Complete code for student Management system using Python Tkinter

VIEW CODE

Summary

This article helps us to use the skills and ideas to develop a student management project in python programming language. We have made use of the inbuilt database SQLite present in Pycharm IDE for database connectivity along with python which makes it super easy to perform database connection and perform the operations on it.

Thank You :)