PL/SQL

(SUBJECT CODE: 25CAP- 602)

MINI PROJECT

EduAdmit: A Smart College Admission Platform

Submitted by:

Prachi Pradyusmita Nayak(25MCI10078)

Boddu Mounika (25MCI10070)

in partial fulfilment for the award of the degree of

Master of Computer Application

CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

Chandigarh University
July 2025 - Nov 2025

Submitted to

Dr. Ashaq Hussain Bhat

DECLARATION

I hereby declare that the project titled "**EduAdmit: A Smart College Admission Platform**" is an original work carried out by me under the guidance of **Dr. Ashaq Hussain Bhat**. The project has been completed as part of the academic requirements for MCA.

The work presented in this report has not been submitted elsewhere for any other academic purpose. Any references made to the work of others have been duly acknowledged in the report.

I take full responsibility for any mistakes or errors that may exist in this report.

Student Name: Prachi Pradyusmita Nayak

Roll No: 25MCI10078

(Signature of Guide)

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to everyone who supported me during the development of this project.

Firstly, I would like to thank **Dr. Ashaq Hussain Bhat**, my supervisor, for their valuable guidance, encouragement, and constant support throughout this project. Their insights helped me improve my understanding of the subject and steer the project in the right direction.

I would also like to thank **University Institute of Computing**, for providing me with the resources and environment necessary to complete this project.

I am grateful to my colleagues and friends who offered their help and shared their expertise during this project. Their assistance was invaluable.

Finally, I wish to thank my family for their patience and support throughout the duration of this project.

Name: Prachi Pradyusmita Nayak

Roll no: 25MCI10078

Index

Introduction

In today's era of digital transformation, educational institutions are increasingly shifting toward automated systems to improve administrative efficiency. One of the most crucial aspects of college administration is the admission process, which involves managing vast amounts of student data such as names, courses, contact details, and admission statuses. Traditionally, this process was handled manually through registers or spreadsheets, making it tedious, time-consuming, and prone to errors. To address these challenges, the **A Smart College Admission Platform (College Admission System)** has been developed as an efficient and intelligent software application that automates the management of student admission records and streamlines institutional workflows.

The **A Smart College Admission Platform (College Admission System)** is built using **Python**, leveraging the **Tkinter** library for the graphical user interface (GUI) and **SQLite3** for secure and lightweight data storage. This combination allows the creation of an interactive, user-friendly application capable of handling complex data operations with minimal effort. The system enables administrators to perform essential functions such as adding new student entries, updating admission statuses, searching existing records, and deleting outdated data. By automating these operations, the software eliminates repetitive manual work, reduces the risk of data duplication, and ensures accurate and reliable record management. Each student record contains vital details including the student's name, roll number, course, email, phone number, and admission status, providing a complete digital profile in a structured format.

The main objective of this project is to provide a **digital solution for admission management**, ensuring that educational institutions can handle student data efficiently and securely. The system not only increases administrative productivity but also minimizes human intervention, ensuring smoother operation of the admission process. Its modular design makes it easy to maintain and upgrade, allowing for future enhancements such as login authentication, cloud-based storage, and report generation. Overall, the **College Admission System** bridges the gap between traditional manual record-keeping and modern digital data management, providing a fast, organized, and scalable platform for educational institutions to manage student admissions effectively.

Objectives

- Automate the process of recording student admission data.
- Provide a simple, interactive GUI for administrators.
- Enable searching and filtering of student records.
- Allow updating of admission statuses (Pending, Approved, Rejected).
- Maintain a persistent database using SQLite3.
- Support error handling, data validation, and user-friendly feedback.

Technologies Used

Component	Technology
Programming Language	Python 3.x
GUI Framework	Tkinter
Database	SQLite3
Modules Used	sqlite3, tkinter, ttk, messagebox
IDE Used	Visual Studio Code / PyCharm
Operating System	Windows / Linux

These technologies together create a lightweight, cross-platform desktop application.

Project Overview

The College Admission System allows users to manage admission records through a graphical interface. It offers CRUD (Create, Read, Update, Delete) functionality and search features.

Each student record includes details such as:

- Name
- Roll Number
- Course
- Email
- Phone Number
- Admission Status

The system prevents duplication of student entries by using **unique name constraints**. It also allows administrators to update the admission status (e.g., from Pending to Approved or Rejected), delete records, or view all students.

System Functionality Overview

The application provides the following core features:

- **Add Record:** Inserts a new student admission entry into the database.
- **Search Record:** Searches by name or roll number.
- **Update Status:** Updates admission status of a selected record.
- **Delete Record:** Removes unwanted entries after confirmation.
- **Show All:** Displays the entire student list in a Tree view table.
- **Clear Fields:** Resets all input fields for new data entry.

A responsive and color-coded GUI enhances usability for administrators.

Algorithm Theory

6.1 Record Management Algorithm

Purpose: To add, update, and delete admission records.
Steps:

1. Capture input values from entry fields.
2. Validate mandatory fields (name, roll number, course).
3. Insert or update the record in the SQLite database.
4. Commit changes and refresh the data table.

6.2 Search & Update Algorithm

Purpose: To filter and modify records dynamically.
Steps:

1. Accept search input from the user.
2. Execute SQL SELECT with LIKE filters.
3. Display matched results in the Tree view widget.
4. On selection, update status or delete record.

7.System Architecture

The project follows a three-tier architecture:

- **Presentation Layer (Frontend):**
  Built with Tkinter, this layer handles data input, button clicks, and visual display of results.
- **Application Layer (Logic):**
  Contains Python functions that process user input, run SQL commands, and manage GUI updates.
- **Data Layer (Backend):**
  SQLite3 database that permanently stores admission records.

Data Flow:

User Input → GUI → SQL Query Execution → Database → Display on GUI Table

8.Source Code Explanation

The project contains key Python functions:

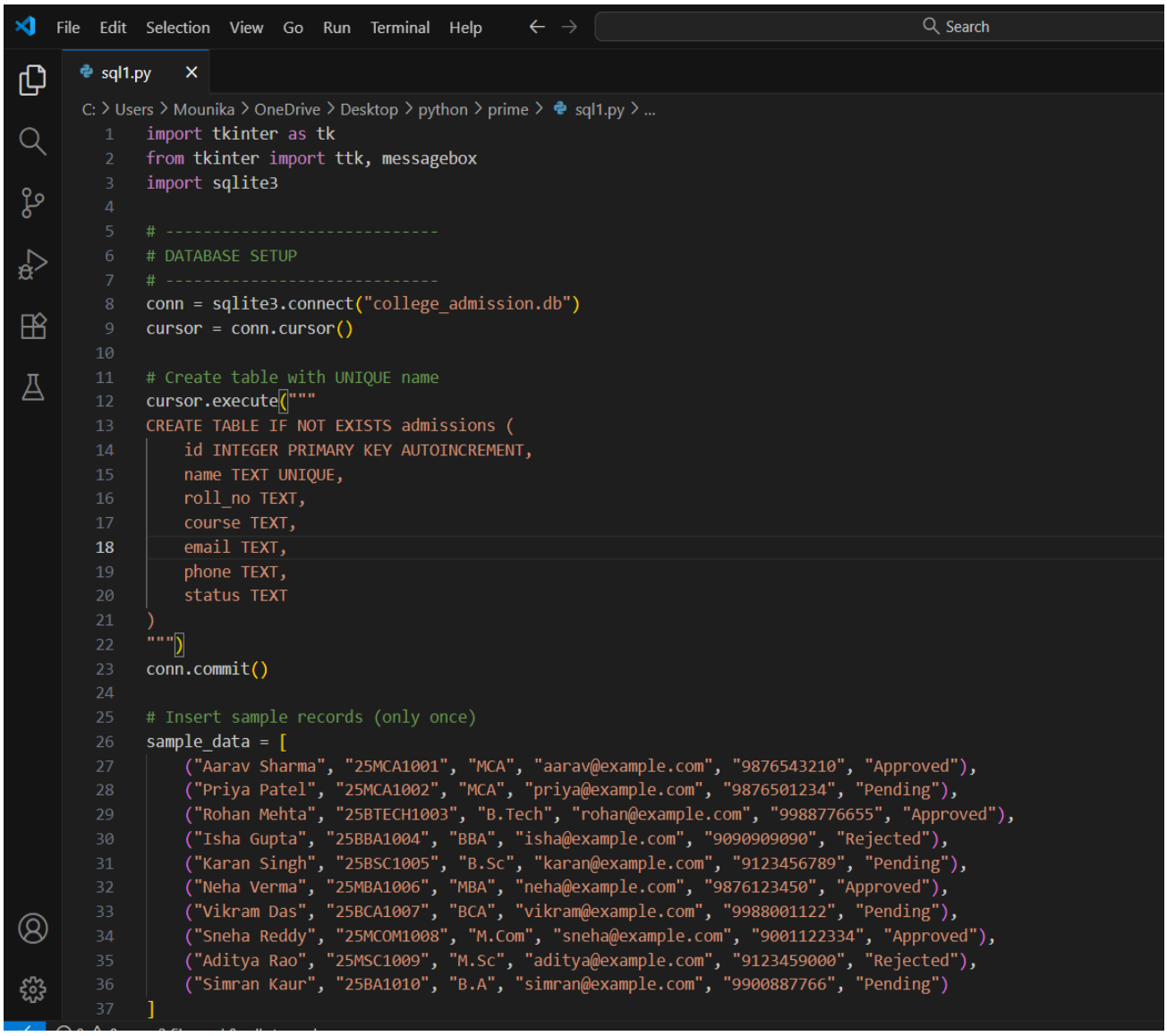
Key Functions:

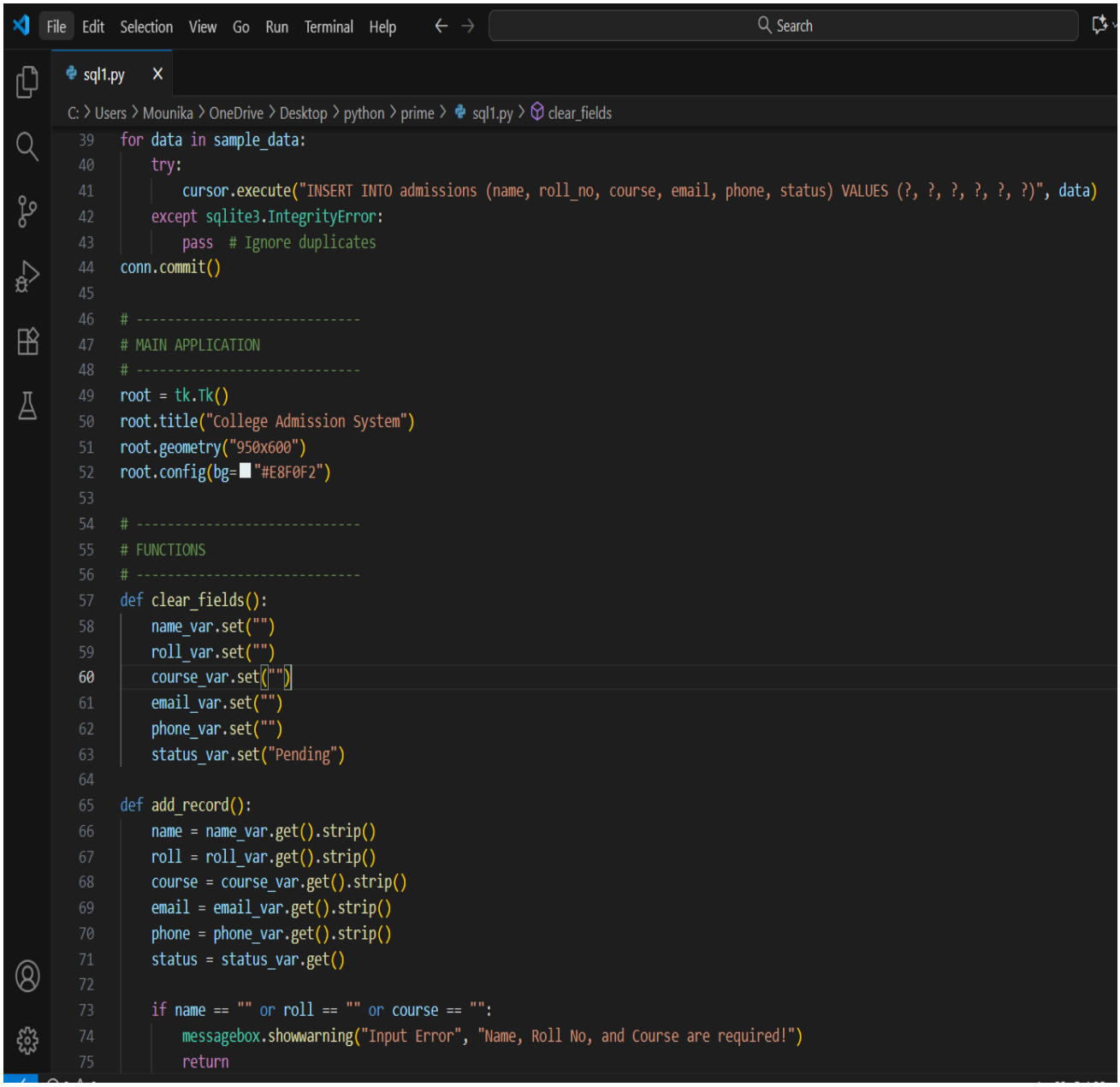
- **add_record():** Adds a new admission entry.
- **update_status():** Updates the selected student's admission status.
- **search_record():** Searches students by name or roll number.
- **delete_record():** Removes a record permanently.
- **fetch_data():** Retrieves all rows from the database for display.
- **clear_fields():** Resets input fields to default.

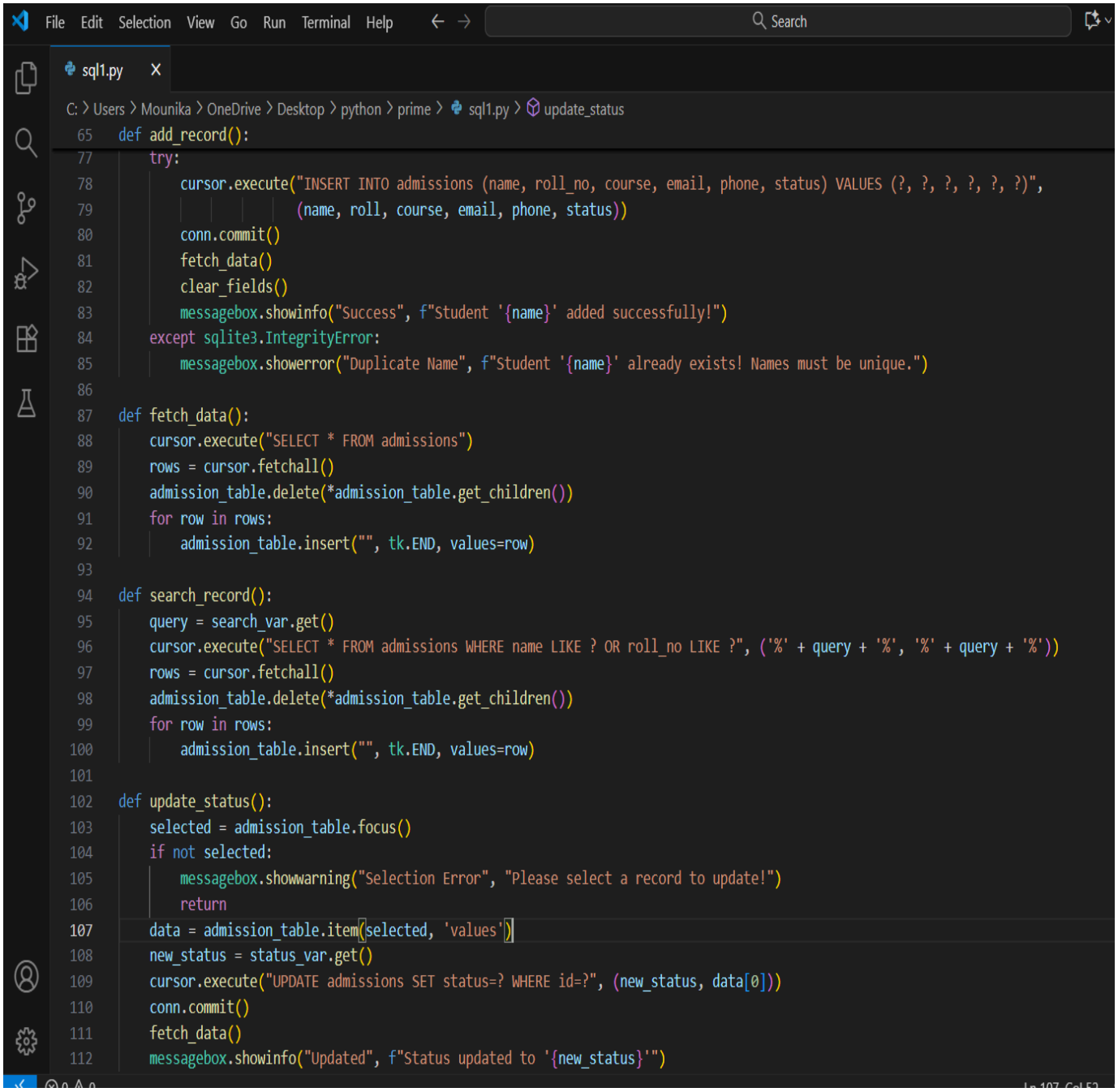
GUI Components:

- Tkinter Frames for layout organization.
- Entry widgets and Combo box for inputs.
- Tree view widget for displaying the database records.
- Buttons linked to CRUD operations.

Code:


File Edit Selection View Go Run Terminal Help
Search
sql1.py
C: > Users > Mounika > OneDrive > Desktop > python > prime > sql1.py > ...
import tkinter as tk
from tkinter import ttk, messagebox
import sqlite3
# ----------------------------
# DATABASE SETUP
# ----------------------------
conn = sqlite3.connect("college_admission.db")
cursor = conn.cursor()
# Create table with UNIQUE name
cursor.execute("""
CREATE TABLE IF NOT EXISTS admissions (
id INTEGER PRIMARY KEY AUTOINCREMENT,
name TEXT UNIQUE,
roll_no TEXT,
course TEXT,
email TEXT,
phone TEXT,
status TEXT
)
""")
conn.commit()
# Insert sample records (only once)
sample_data = [
("Aarav Sharma", "25MCA1001", "MCA", "aarav@example.com", "9876543210", "Approved"),
("Priya Patel", "25MCA1002", "MCA", "priya@example.com", "9876501234", "Pending"),
("Rohan Mehta", "25BTECH1003", "B.Tech", "rohan@example.com", "9988776655", "Approved"),
("Isha Gupta", "25BBA1004", "BBA", "isha@example.com", "9090909090", "Rejected"),
("Karan Singh", "25BSC1005", "B.Sc", "karan@example.com", "9123456789", "Pending"),
("Neha Verma", "25MBA1006", "MBA", "neha@example.com", "9876123450", "Approved"),
("Vikram Das", "25BCA1007", "BCA", "vikram@example.com", "9988001122", "Pending"),
("Sneha Reddy", "25MCOM1008", "M.Com", "sneha@example.com", "9001122334", "Approved"),
("Aditya Rao", "25MSC1009", "M.Sc", "aditya@example.com", "9123459000", "Rejected"),
("Simran Kaur", "25BA1010", "B.A", "simran@example.com", "9900887766", "Pending")
]


File Edit Selection View Go Run Terminal Help
Search
sql1.py
C: > Users > Mounika > OneDrive > Desktop > python > prime > sql1.py > clear_fields
for data in sample_data:
try:
cursor.execute("INSERT INTO admissions (name, roll_no, course, email, phone, status) VALUES (?, ?, ?, ?, ?, ?)", data)
except sqlite3.IntegrityError:
pass # Ignore duplicates
conn.commit()
# ----------
# MAIN APPLICATION
# ----------
root = tk.Tk()
root.title("College Admission System")
root.geometry("950x600")
root.config(bg="#E8F0F2")
# ----------
# FUNCTIONS
# ----------
def clear_fields():
name_var.set("")
roll_var.set("")
course_var.set("")
email_var.set("")
phone_var.set("")
status_var.set("Pending")
def add_record():
name = name_var.get().strip()
roll = roll_var.get().strip()
course = course_var.get().strip()
email = email_var.get().strip()
phone = phone_var.get().strip()
status = status_var.get()
if name == "" or roll == "" or course == "":
messagebox.showwarning("Input Error", "Name, Roll No, and Course are required!")
return



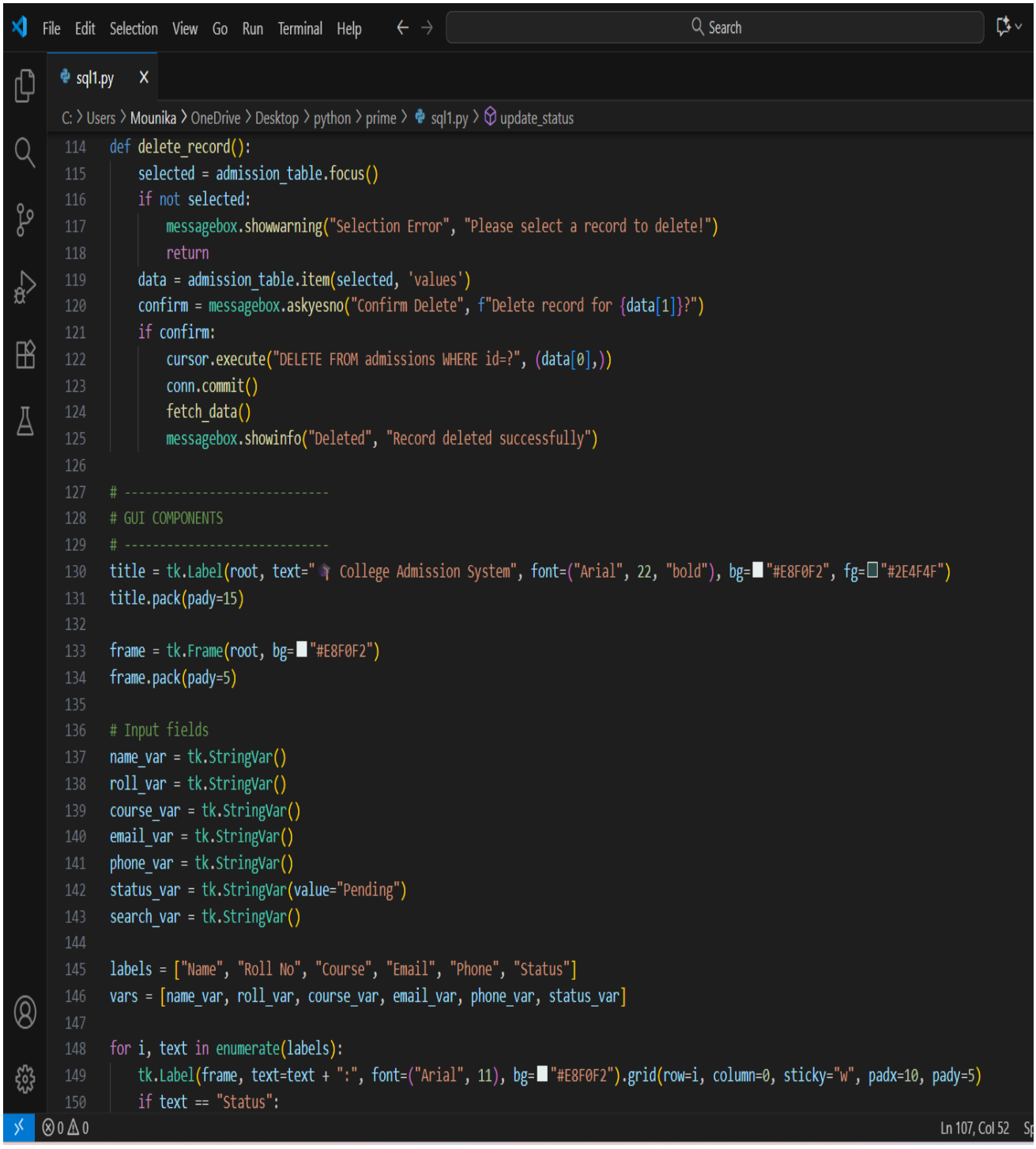
File Edit Selection View Go Run Terminal Help

Search

sql1.py

C: > Users > Mounika > OneDrive > Desktop > python > prime > sql1.py > update_status

```
65  def add_record():
77      try:
78          cursor.execute("INSERT INTO admissions (name, roll_no, course, email, phone, status) VALUES (?, ?, ?, ?, ?, ?)",
79                         (name, roll, course, email, phone, status))
80          conn.commit()
81          fetch_data()
82          clear_fields()
83          messagebox.showinfo("Success", f"Student '{name}' added successfully!")
84      except sqlite3.IntegrityError:
85          messagebox.showerror("Duplicate Name", f"Student '{name}' already exists! Names must be unique.")
86
87  def fetch_data():
88      cursor.execute("SELECT * FROM admissions")
89      rows = cursor.fetchall()
90      admission_table.delete(*admission_table.get_children())
91      for row in rows:
92          admission_table.insert("", tk.END, values=row)
93
94  def search_record():
95      query = search_var.get()
96      cursor.execute("SELECT * FROM admissions WHERE name LIKE ? OR roll_no LIKE ?", ('%' + query + '%', '%' + query + '%'))
97      rows = cursor.fetchall()
98      admission_table.delete(*admission_table.get_children())
99      for row in rows:
100         admission_table.insert("", tk.END, values=row)
101
102 def update_status():
103     selected = admission_table.focus()
104     if not selected:
105         messagebox.showwarning("Selection Error", "Please select a record to update!")
106         return
107     data = admission_table.item(selected, 'values')
108     new_status = status_var.get()
109     cursor.execute("UPDATE admissions SET status=? WHERE id=?", (new_status, data[0]))
110     conn.commit()
111     fetch_data()
112     messagebox.showinfo("Updated", f"Status updated to '{new_status}'")
```



File Edit Selection View Go Run Terminal Help

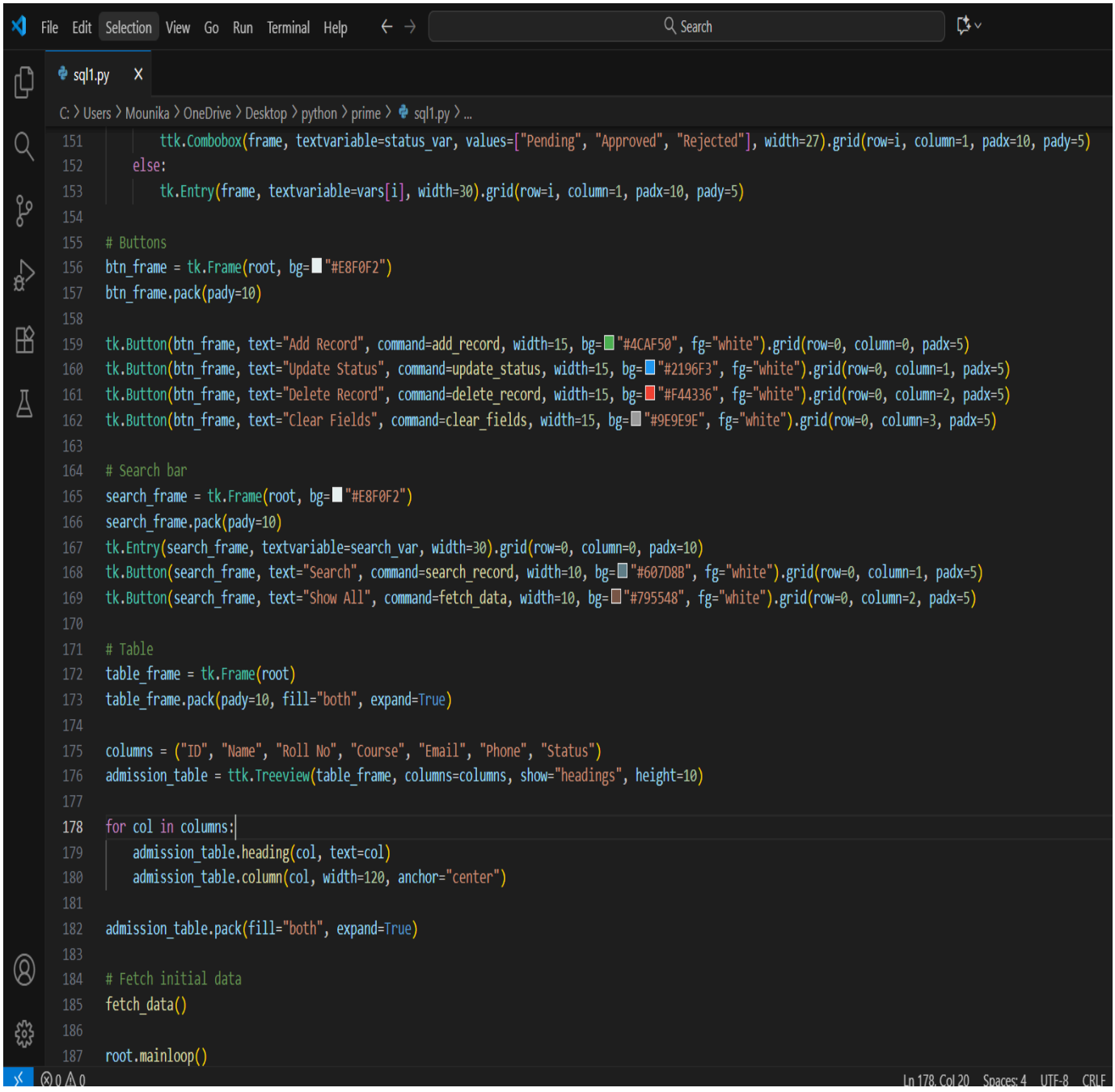
Search

sql1.py

C: > Users > Mounika > OneDrive > Desktop > python > prime > sql1.py > update_status

```python
def delete_record():
    selected = admission_table.focus()
    if not selected:
        messagebox.showwarning("Selection Error", "Please select a record to delete!")
        return
    data = admission_table.item(selected, 'values')
    confirm = messagebox.askyesno("Confirm Delete", f"Delete record for {data[1]}?")
    if confirm:
        cursor.execute("DELETE FROM admissions WHERE id=?", (data[0],))
        conn.commit()
        fetch_data()
        messagebox.showinfo("Deleted", "Record deleted successfully")

# ----------------------------
# GUI COMPONENTS
# ----------------------------
title = tk.Label(root, text="🎓 College Admission System", font=("Arial", 22, "bold"), bg="#E8F0F2", fg="#2E4F4F")
title.pack(pady=15)

frame = tk.Frame(root, bg="#E8F0F2")
frame.pack(pady=5)

# Input fields
name_var = tk.StringVar()
roll_var = tk.StringVar()
course_var = tk.StringVar()
email_var = tk.StringVar()
phone_var = tk.StringVar()
status_var = tk.StringVar(value="Pending")
search_var = tk.StringVar()

labels = ["Name", "Roll No", "Course", "Email", "Phone", "Status"]
vars = [name_var, roll_var, course_var, email_var, phone_var, status_var]

for i, text in enumerate(labels):
    tk.Label(frame, text=text + ":", font=("Arial", 11), bg="#E8F0F2").grid(row=i, column=0, sticky="w", padx=10, pady=5)
    if text == "Status":
```

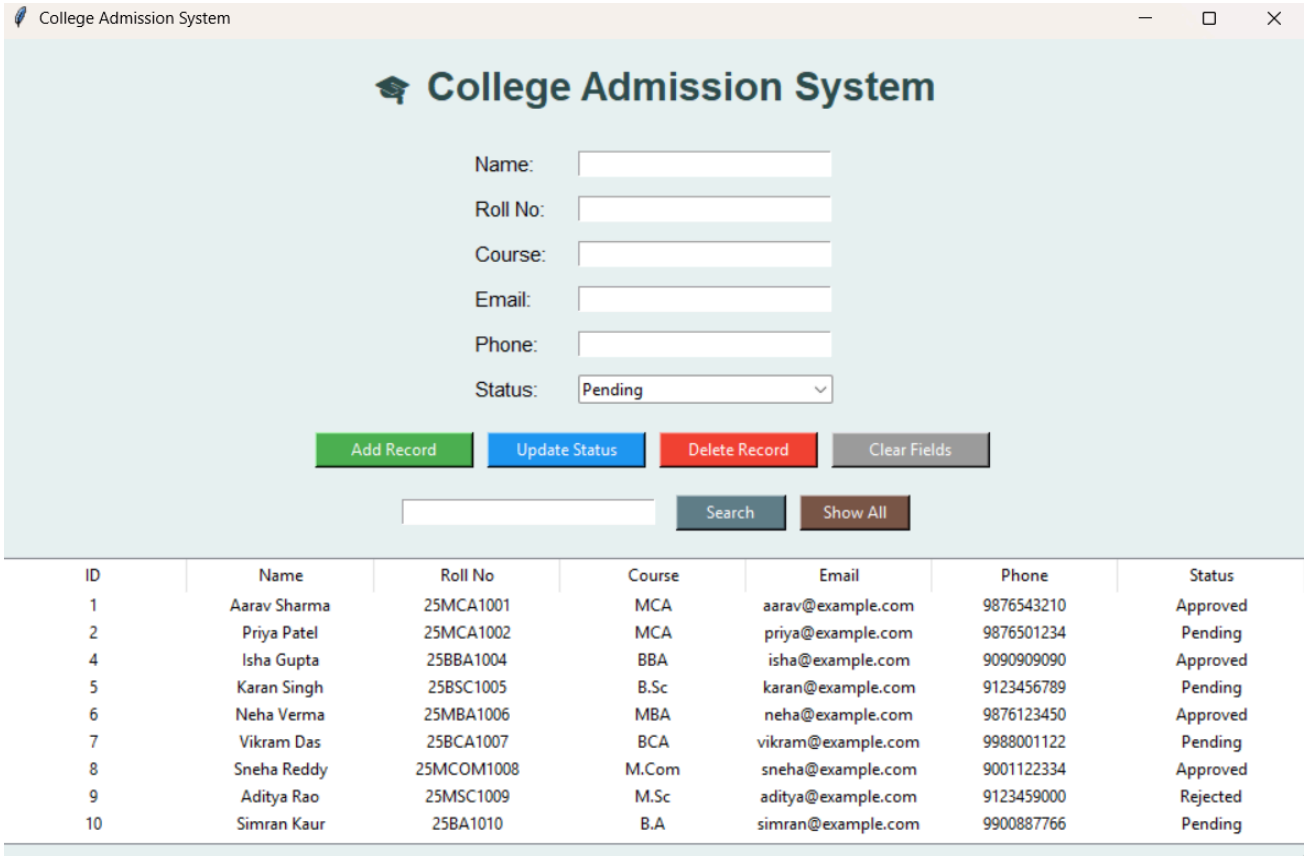
⊗0 ⚠0 Ln 107, Col 52



```python
            ttk.Combobox(frame, textvariable=status_var, values=["Pending", "Approved", "Rejected"], width=27).grid(row=i, column=1, padx=10, pady=5)
        else:
            tk.Entry(frame, textvariable=vars[i], width=30).grid(row=i, column=1, padx=10, pady=5)

# Buttons
btn_frame = tk.Frame(root, bg="#E8F0F2")
btn_frame.pack(pady=10)

tk.Button(btn_frame, text="Add Record", command=add_record, width=15, bg="#4CAF50", fg="white").grid(row=0, column=0, padx=5)
tk.Button(btn_frame, text="Update Status", command=update_status, width=15, bg="#2196F3", fg="white").grid(row=0, column=1, padx=5)
tk.Button(btn_frame, text="Delete Record", command=delete_record, width=15, bg="#F44336", fg="white").grid(row=0, column=2, padx=5)
tk.Button(btn_frame, text="Clear Fields", command=clear_fields, width=15, bg="#9E9E9E", fg="white").grid(row=0, column=3, padx=5)

# Search bar
search_frame = tk.Frame(root, bg="#E8F0F2")
search_frame.pack(pady=10)
tk.Entry(search_frame, textvariable=search_var, width=30).grid(row=0, column=0, padx=10)
tk.Button(search_frame, text="Search", command=search_record, width=10, bg="#607D8B", fg="white").grid(row=0, column=1, padx=5)
tk.Button(search_frame, text="Show All", command=fetch_data, width=10, bg="#795548", fg="white").grid(row=0, column=2, padx=5)

# Table
table_frame = tk.Frame(root)
table_frame.pack(pady=10, fill="both", expand=True)

columns = ("ID", "Name", "Roll No", "Course", "Email", "Phone", "Status")
admission_table = ttk.Treeview(table_frame, columns=columns, show="headings", height=10)

for col in columns:
    admission_table.heading(col, text=col)
    admission_table.column(col, width=120, anchor="center")

admission_table.pack(fill="both", expand=True)

# Fetch initial data
fetch_data()

root.mainloop()
```

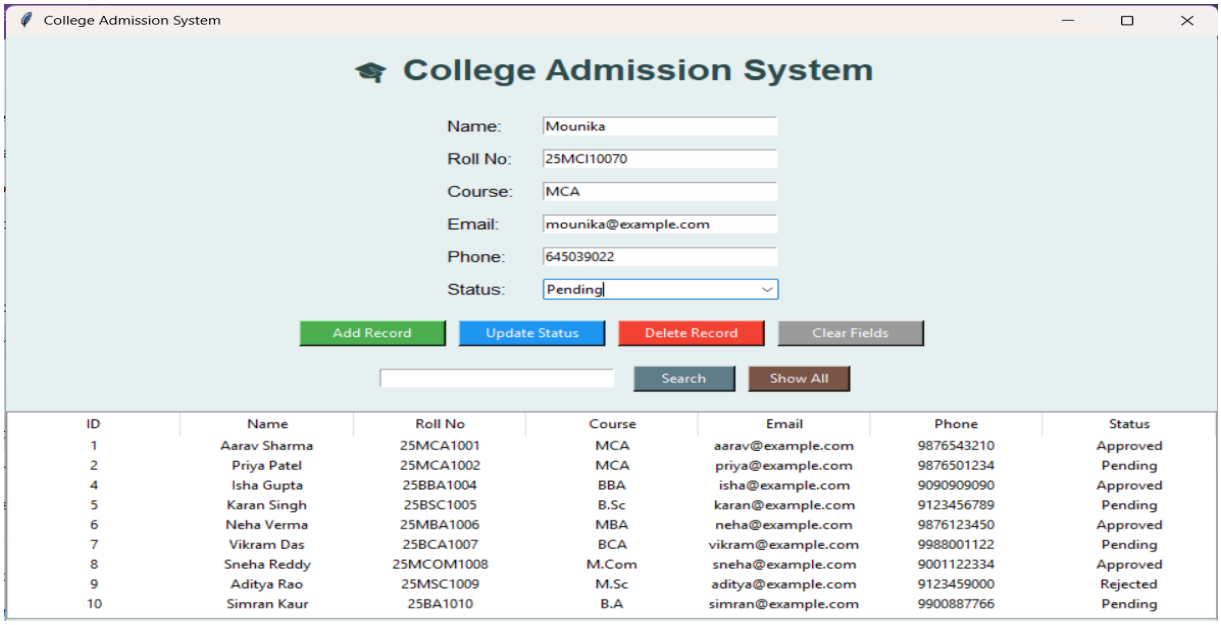
Output Screenshots

Figure 1: Main Application Interface



ID	Name	Roll No	Course	Email	Phone	Status
1	Aarav Sharma	25MCA1001	MCA	aarav@example.com	9876543210	Approved
2	Priya Patel	25MCA1002	MCA	priya@example.com	9876501234	Pending
4	Isha Gupta	25BBA1004	BBA	isha@example.com	9090909090	Approved
5	Karan Singh	25BSC1005	B.Sc	karan@example.com	9123456789	Pending
6	Neha Verma	25MBA1006	MBA	neha@example.com	9876123450	Approved
7	Vikram Das	25BCA1007	BCA	vikram@example.com	9988001122	Pending
8	Sneha Reddy	25MCOM1008	M.Com	sneha@example.com	9001122334	Approved
9	Aditya Rao	25MSC1009	M.Sc	aditya@example.com	9123459000	Rejected
10	Simran Kaur	25BA1010	B.A	simran@example.com	9900887766	Pending

Figure 2 and 3: Add Record Form



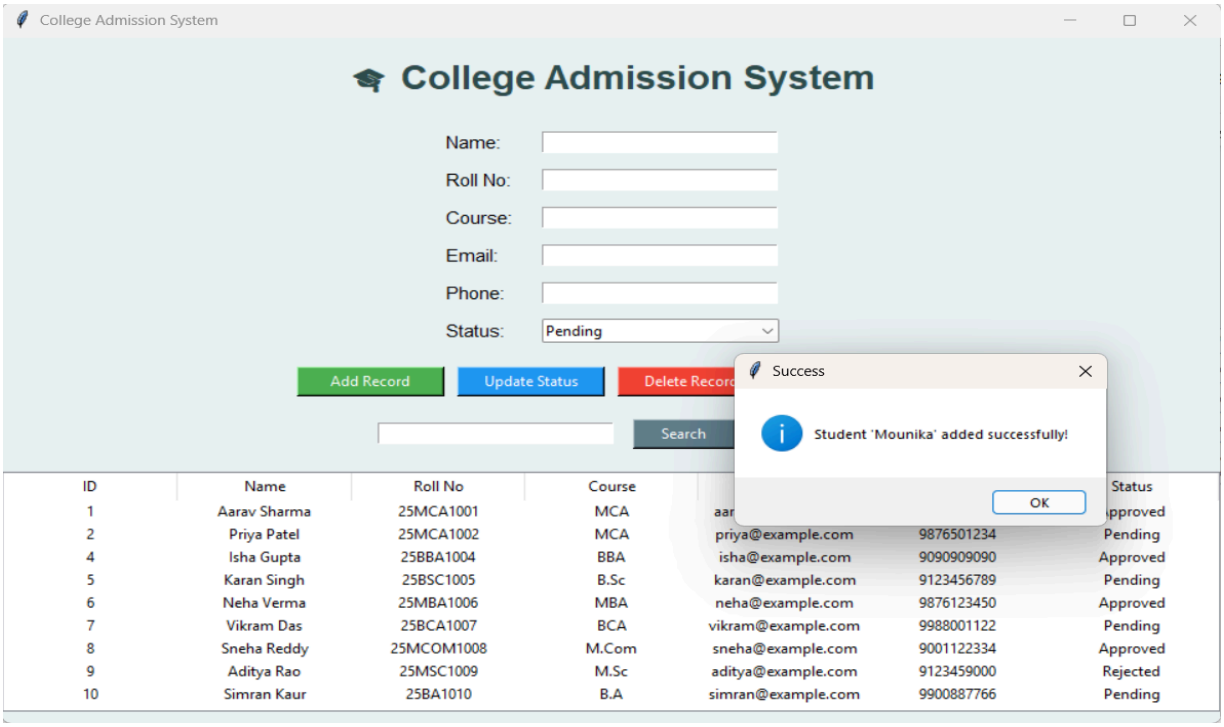
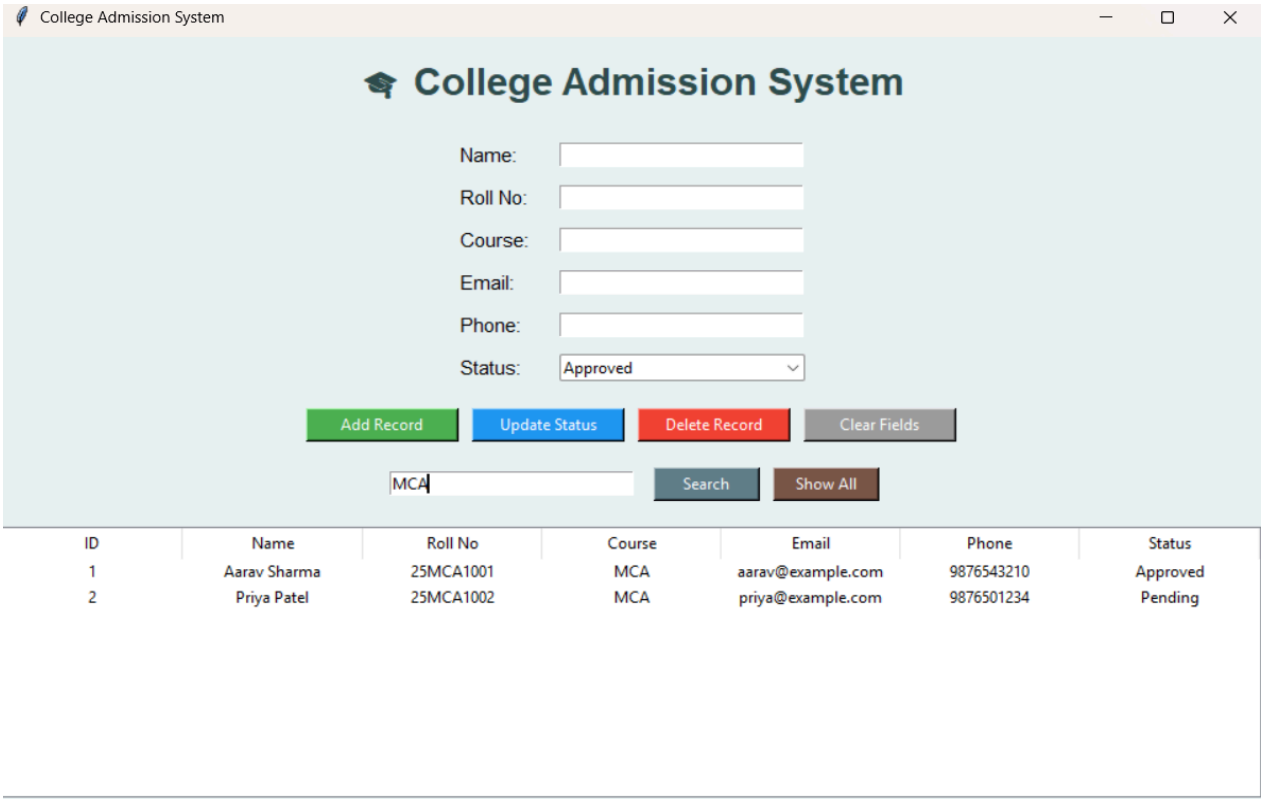


Figure 4 and 5: Search & Update Window



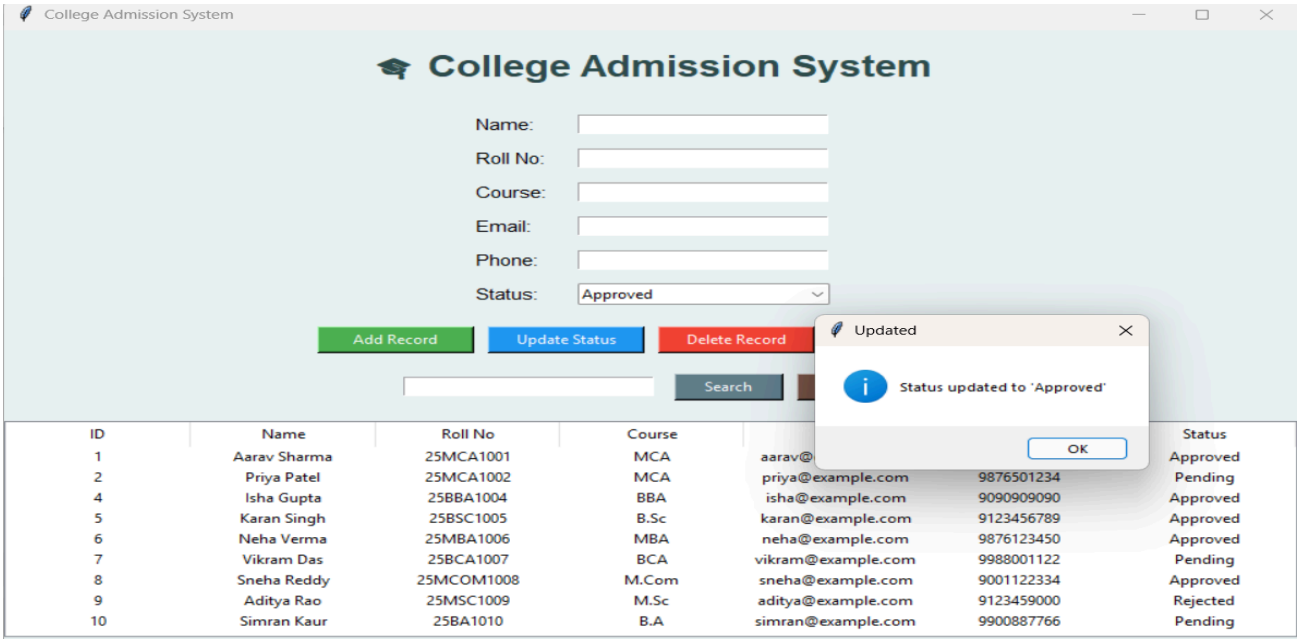
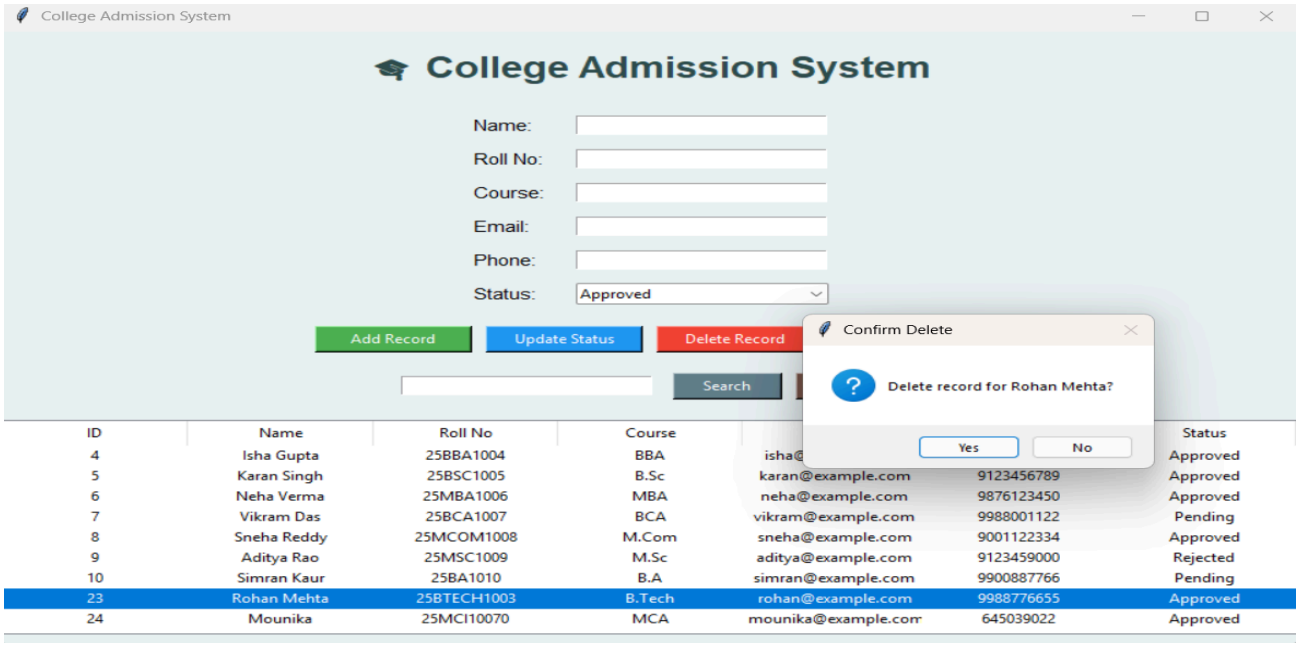


Figure 6: Delete the record



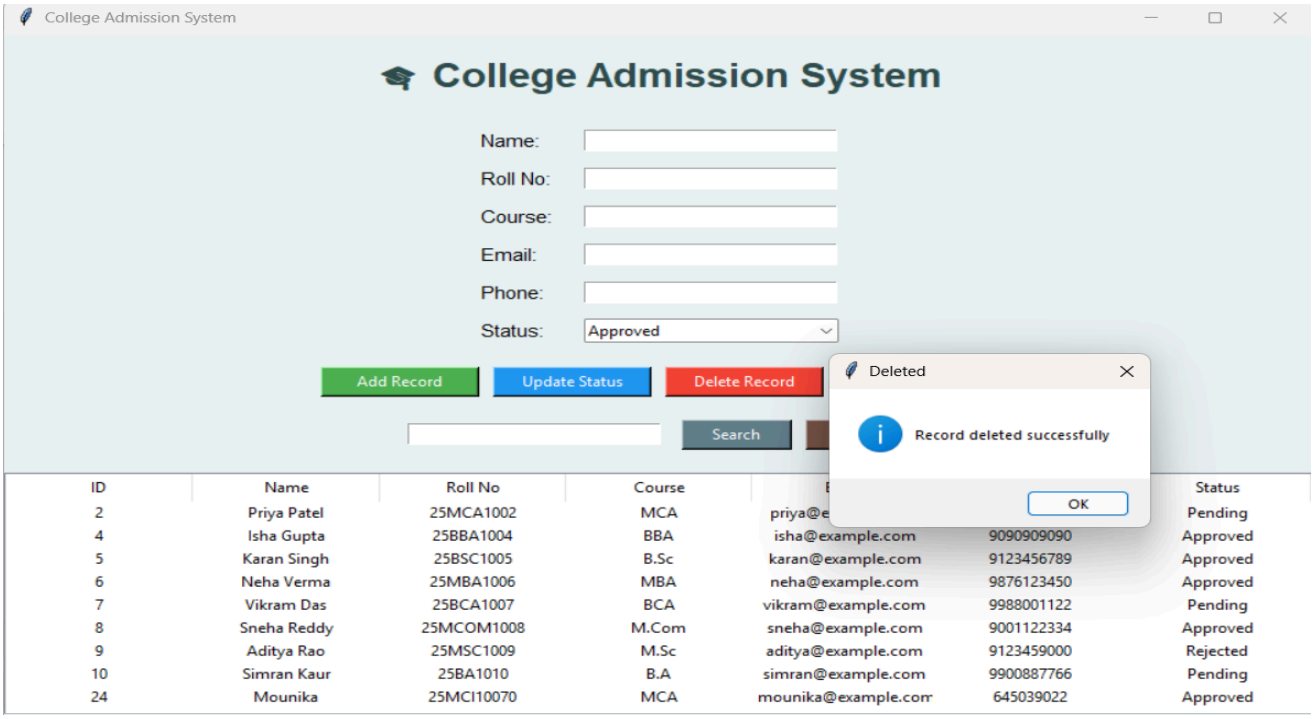
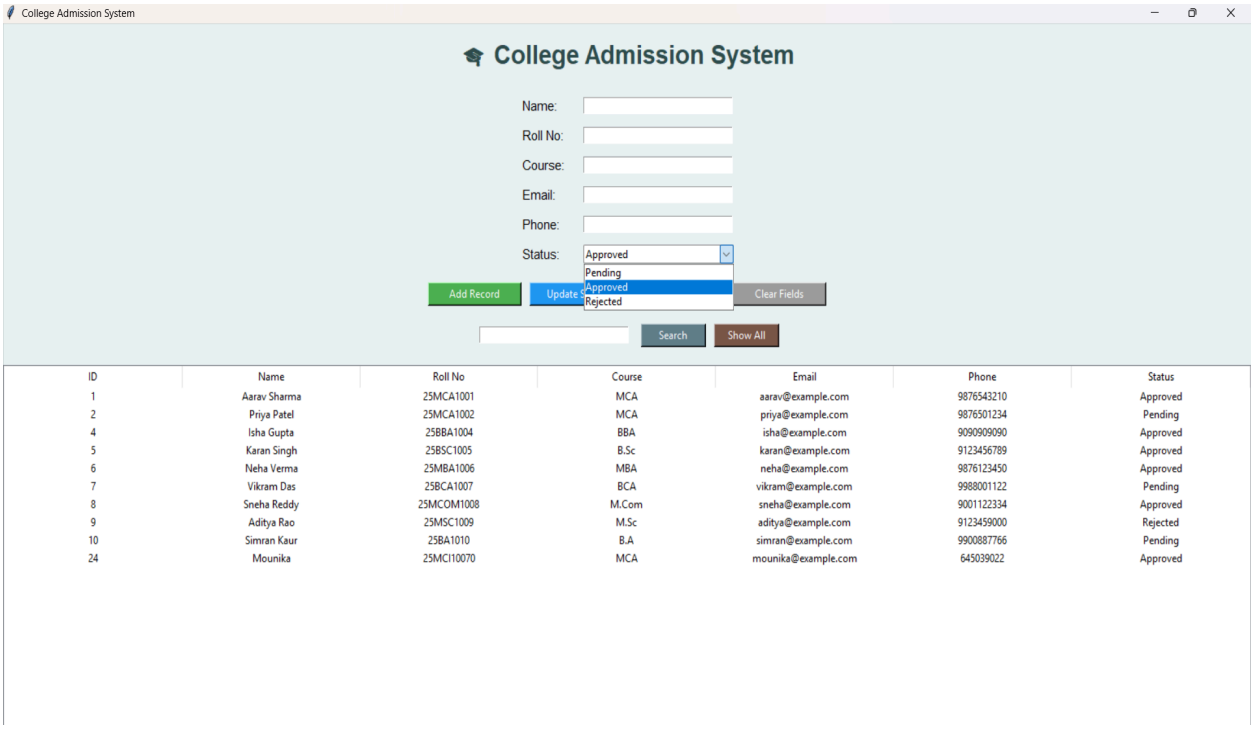


Figure 7: Admission Status Table Display



Result

The system was successfully executed with the following outcomes:

- Admission records were stored and retrieved efficiently.
- Status updates were reflected in real time.
- Duplicate entries were restricted using unique name validation.
- GUI was clean, fast, and interactive.

Conclusion

The **College Admission System** effectively demonstrates how modern programming tools can be utilized to simplify and automate administrative processes in educational institutions. By integrating **Python**, **Tkinter**, and **SQLite3**, the project successfully replaces traditional, manual methods of managing student admissions with a more structured and efficient digital approach. The system allows administrators to seamlessly add, update, delete, and search student records, thereby reducing manual errors, saving time, and improving overall operational productivity. Through its intuitive graphical interface, the application provides a smooth user experience and ensures that all data remains organized and accessible at all times.

This project highlights the practical implementation of database connectivity, GUI design, and logical data handling within a single application. It not only fulfils its objective of managing admission data efficiently but also provides a strong foundation for the development of more advanced institutional management systems. The successful integration of front-end and back-end components in this project demonstrates the potential of Python as a versatile language for building real-world, data-driven applications. Overall, the **College Admission System** represents an important step toward digital transformation in education, promoting accuracy, efficiency, and eco-friendly, paperless record management.

Future Scope

1. Integrate login authentication for admin access.
2. Export data to Excel or PDF for record keeping.
3. Add data visualization (e.g., number of admissions by course).
4. Implement cloud-based database storage.
5. Provide multi-user access over a network.
6. Enable automatic email or SMS notifications for admission updates.
7. Add search filters for courses, status, and academic year.
8. Include data backup and restore functionality for safety.
9. Develop a mobile or web-based interface for remote access.
10. Integrate AI tools to analyze admission trends and predict future enrollments

References

1. Lundh, F. (2005). Python Standard Library. O'Reilly Media.
2. Grayson, J. E. (2000). Python and Tkinter Programming. Manning Publications.
3. Van Rossum, G., & Drake, F. L. (2009). Python 3 Reference Manual. CreateSpace.
4. SQLite3 Documentation (2024). SQLite Database Management.
5. Tkinter Official Documentation (2024). GUI Programming Guide.