VIT BHOPAL

PROJECT REPORT

Faculty Data CRUD Manager

COURSE

PYTHON ESSENTIALS

NIYATI SETHI
25BCE10997

Table of Contents

Introduction

Managing basic faculty information such as names, departments, qualifications, and experience is a common requirement in academic settings. When this information is maintained manually in notebooks or scattered across spreadsheets, updating and retrieving it becomes inconvenient. There is also a higher chance of errors or outdated entries.

This project aims to develop a small but functional desktop application to simplify faculty data management. The system allows users to add new faculty records, view all existing entries in a table, update any selected record, and delete entries that are no longer required. It has been developed using Python, Tkinter for the user interface, and SQLite for database storage.

The purpose of the project is not only to build a working CRUD application but also to understand basic programming concepts, GUI development, data handling, and modular code structure. Through this project, I was able to explore how different components of a simple application come together to form a complete solution.

Problem Statement

Problem

Handling faculty information manually can lead to several issues such as inconsistent data, difficulty in locating records, and challenges in updating old entries. When information is scattered across different files or handwritten documents, maintaining it becomes time-consuming and prone to mistakes.

Vision

To address this, there is a need for a simple system where:

- New faculty details can be added easily
- All stored entries can be viewed together
- Any selected entry can be modified without hassle
- Old or incorrect records can be deleted
- The data remains stored permanently in a structured format

SCOPE

1. **Basic Faculty Management System**
   This project focuses on building a simple desktop application that allows faculty information to be stored and managed in an organized digital format, reducing dependence on manual or spreadsheet-based methods.

2. **Core CRUD Functionality**
   The scope includes implementing essential operations such as adding new faculty details, viewing the stored list, updating existing records, and deleting inaccurate or unwanted entries.

3. **Local Storage Using SQLite**
   The system is designed for single-user use and relies on SQLite for storing data locally. This keeps the project lightweight and removes the need for server setups or online database systems.

4. **Limited Faculty Data Fields**
   The project handles only basic fields—name, department, qualification, and experience. It intentionally excludes complex academic or administrative data to maintain clarity and simplicity.

5. **Exclusion of Advanced Features**
   Features like login systems, cloud syncing, multi-role access, advanced analytics, and elaborate UI design are beyond the scope. These are intentionally not included to keep the project focused on learning fundamentals.

6. **Learning-Oriented Project Structure**
   The project mainly aims to help understand GUI building, database operations, and modular programming in Python. The system acts as a foundation that can be expanded in the future but remains simple at this stage for educational clarity.

OBJECTIVE

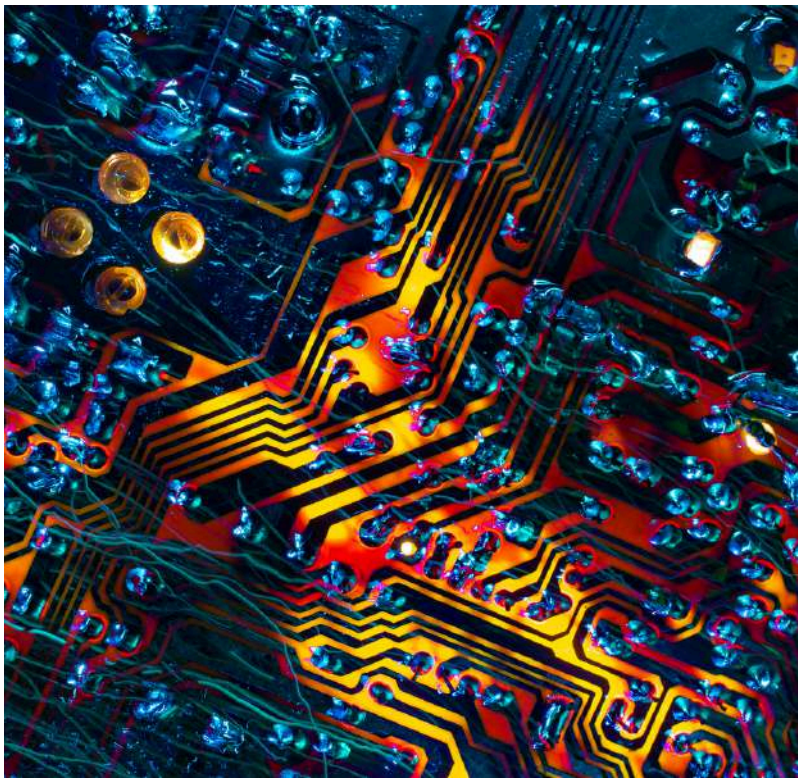


- Build a Functional Management Tool
- Simplify Faculty Record Handling
- Practice CRUD Operations
- Learn GUI Development
- Connect GUI with a Database
- Implement Modular Programming
- Develop Practical Programming Skills

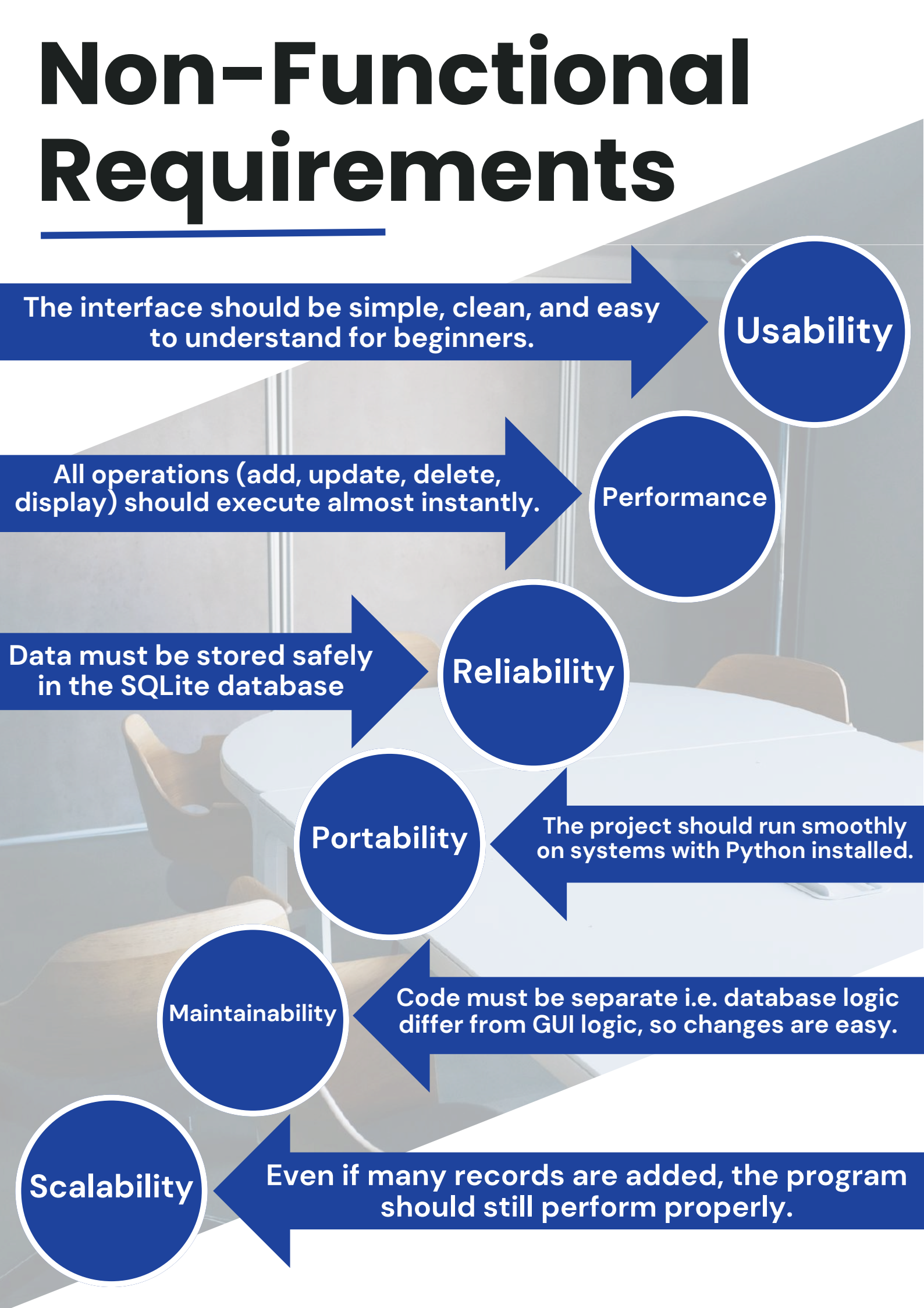
Functional Requirements

These are the specific functions the system must perform:

- The system should allow entering new faculty details (Name, Department, Qualification, Experience).
- It should store the faculty data permanently in a database.
- The system must display all saved records in a table format.
- Users must be able to select any record from the table.
- The system should allow updating the selected record with new values.
- The system should allow deleting any selected record.
- The system should refresh the table automatically after every insert, update, or delete.



Non-Functional Requirements



System Architecture

The system follows a simple three-layer architecture

User Interface Layer (GUI – Tkinter)

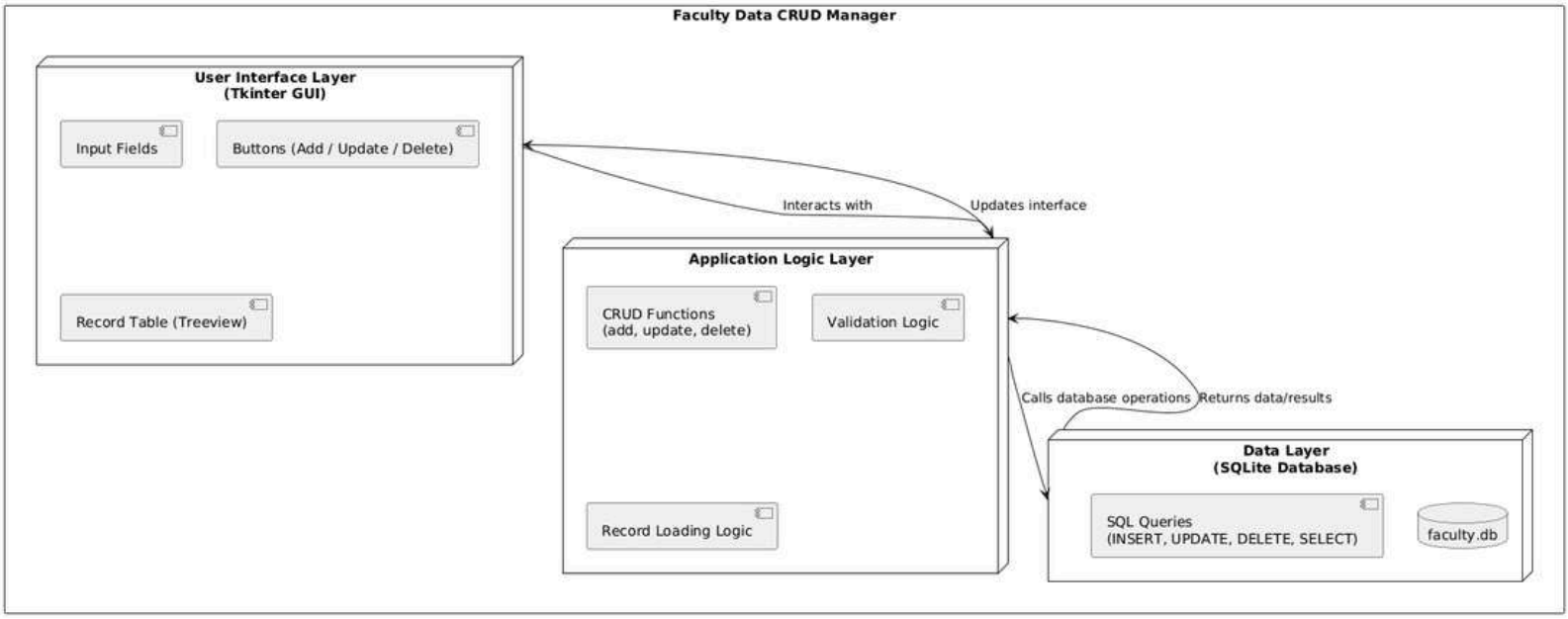
This layer handles all the screens, input boxes, buttons, and the table used to display faculty records.

Logic Layer (Python Functions)

This layer processes inputs (like adding or updating), triggers database functions, and refreshes the displayed data.

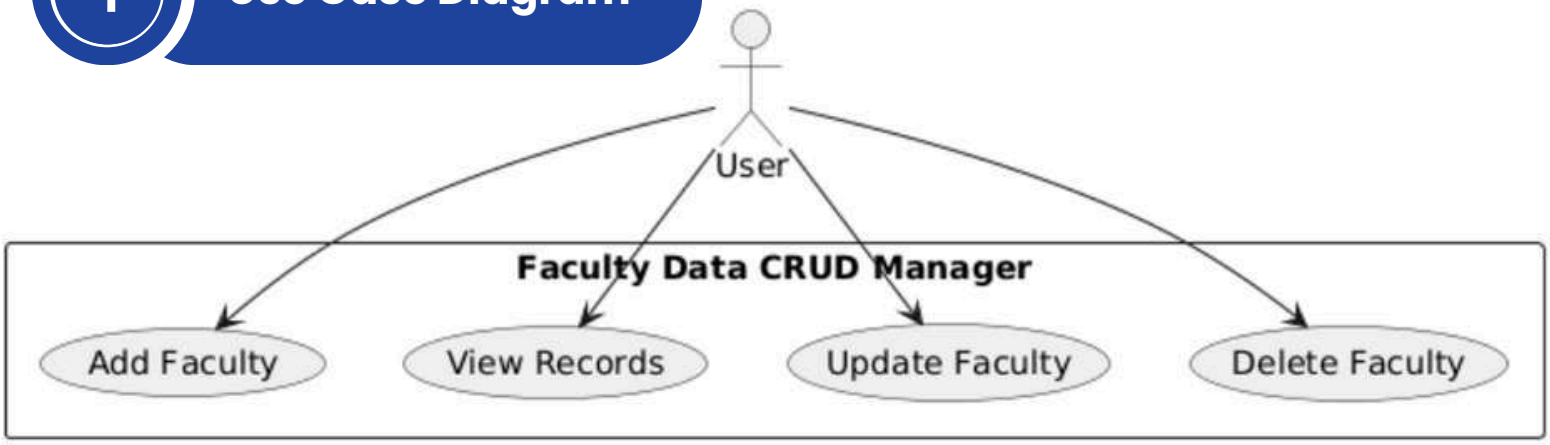
Database Layer (SQLite)

This layer stores the faculty information in a structured table called faculty. It acts as permanent storage for all entries.

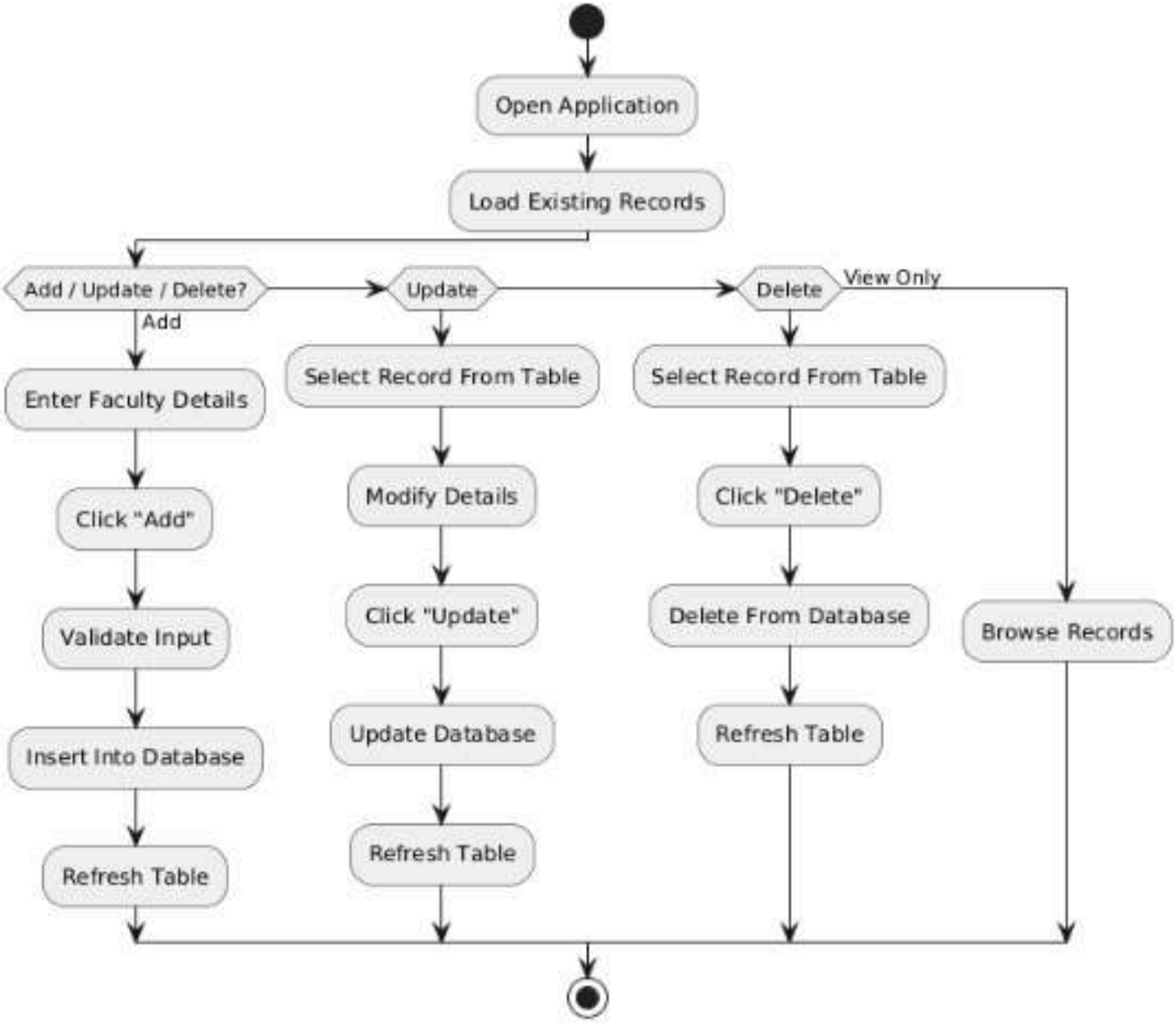


Design DIAGRAMS

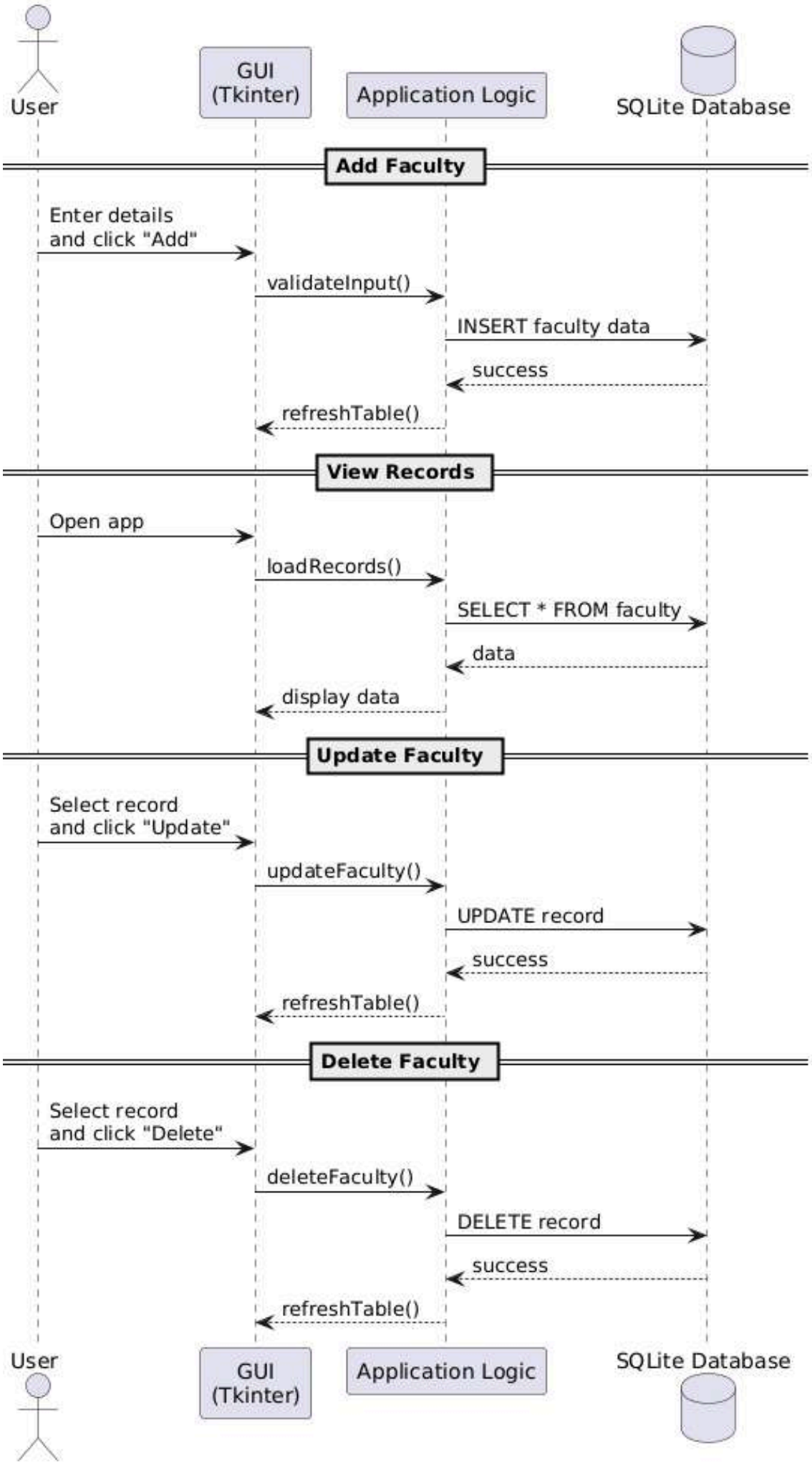
1 Use Case Diagram



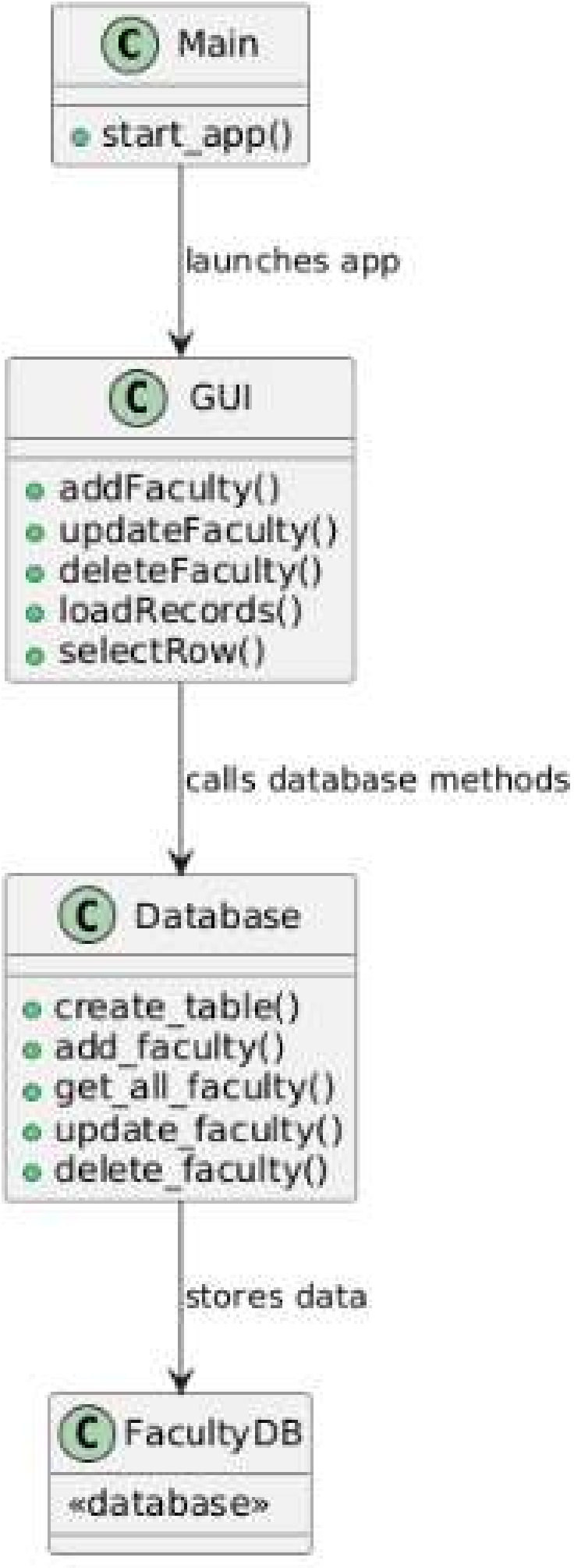
2 Workflow Diagram



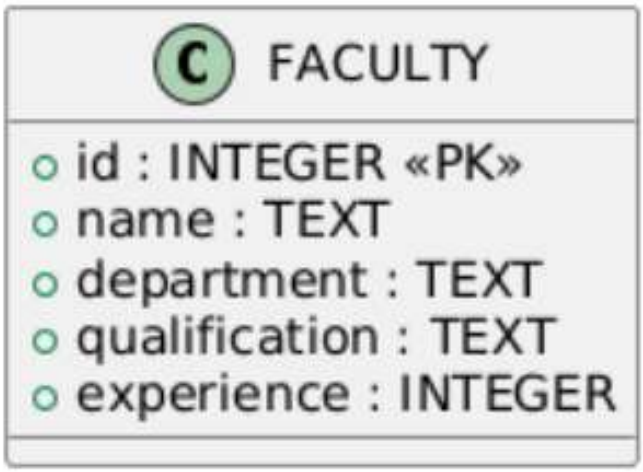
3 Sequence Diagram



4 Class Diagram



5 ER Diagram



Design Decisions & Rationale

1. **Using Tkinter for GUI**
   Tkinter is built into Python, easy to learn, and perfect for small applications. For this project, I didn't need a fancy interface, so Tkinter made sense.

2. **Using SQLite for the database**
   SQLite is lightweight and does not require a server setup.
   A simple .db file is enough for storing faculty data.

3. **Separating GUI and database code**
   I kept the database functions in a separate file (database.py) and the GUI in gui.py.
   This makes the project clearer and easier to maintain.

4. **Using Treeview for the table**
   Treeview provided a clean way to display multiple fields of each record. It also allows easy row selection, which was required for update and delete.

5. **Adding validation messages**
   To avoid wrong or empty inputs, I added basic checks before adding or updating records. This makes the app more user-friendly.

Implementation Details

The project is split into different files, each responsible for a specific part of the system.

gui.py

This contains:

- Input fields
- Buttons
- Treeview table
- Functions for add, update, delete
- Basic validation
- Display messages for errors and successes

It handles everything the user sees and interacts with.

database.py

This file contains all the SQL logic:

- Creating the table
- Adding new records
- Fetching all records
- Updating records
- Deleting records

The GUI calls these functions whenever needed.

main.py

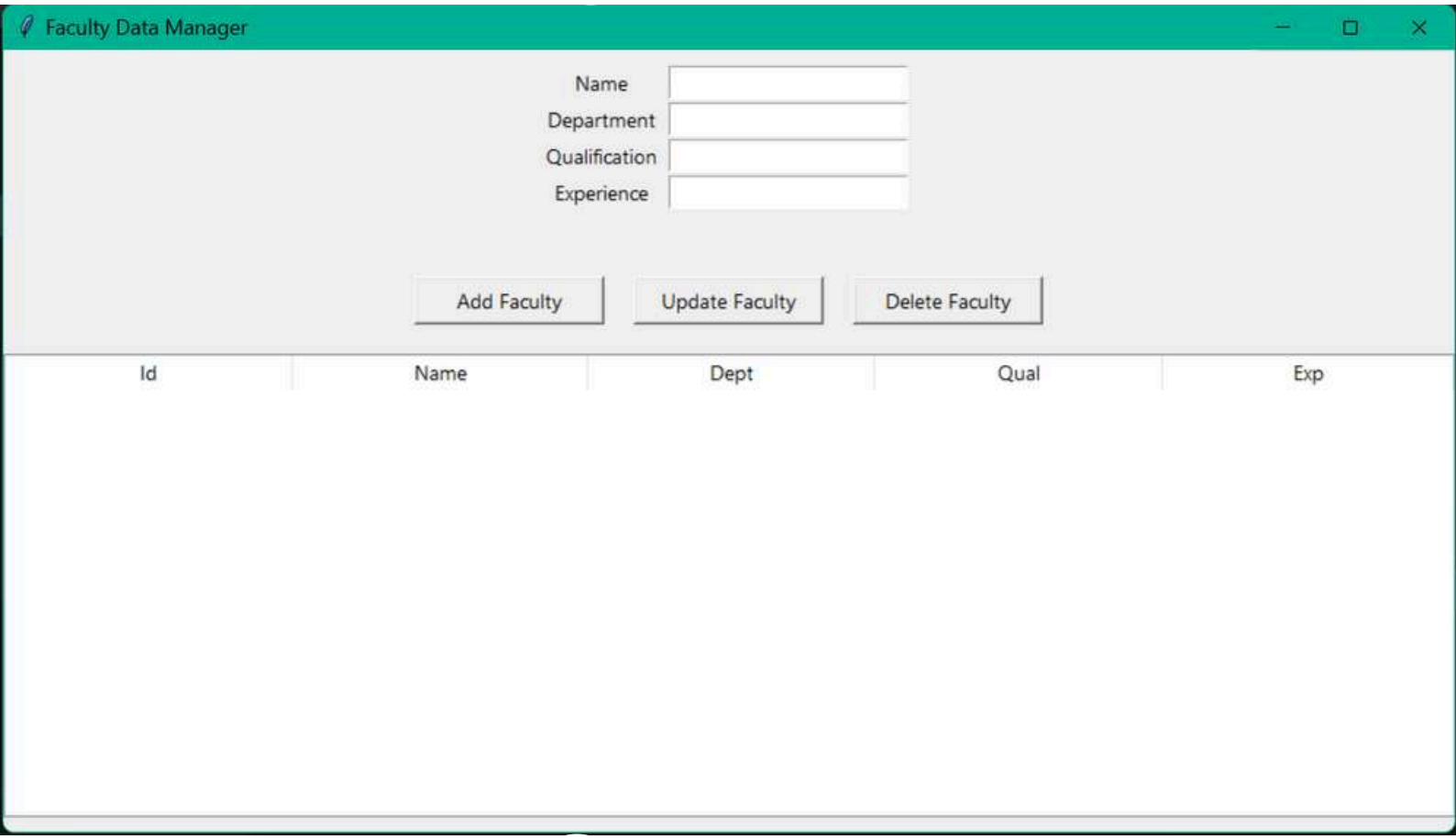
This is the file that simply starts the application by calling the main function from gui.py.

faculty.db

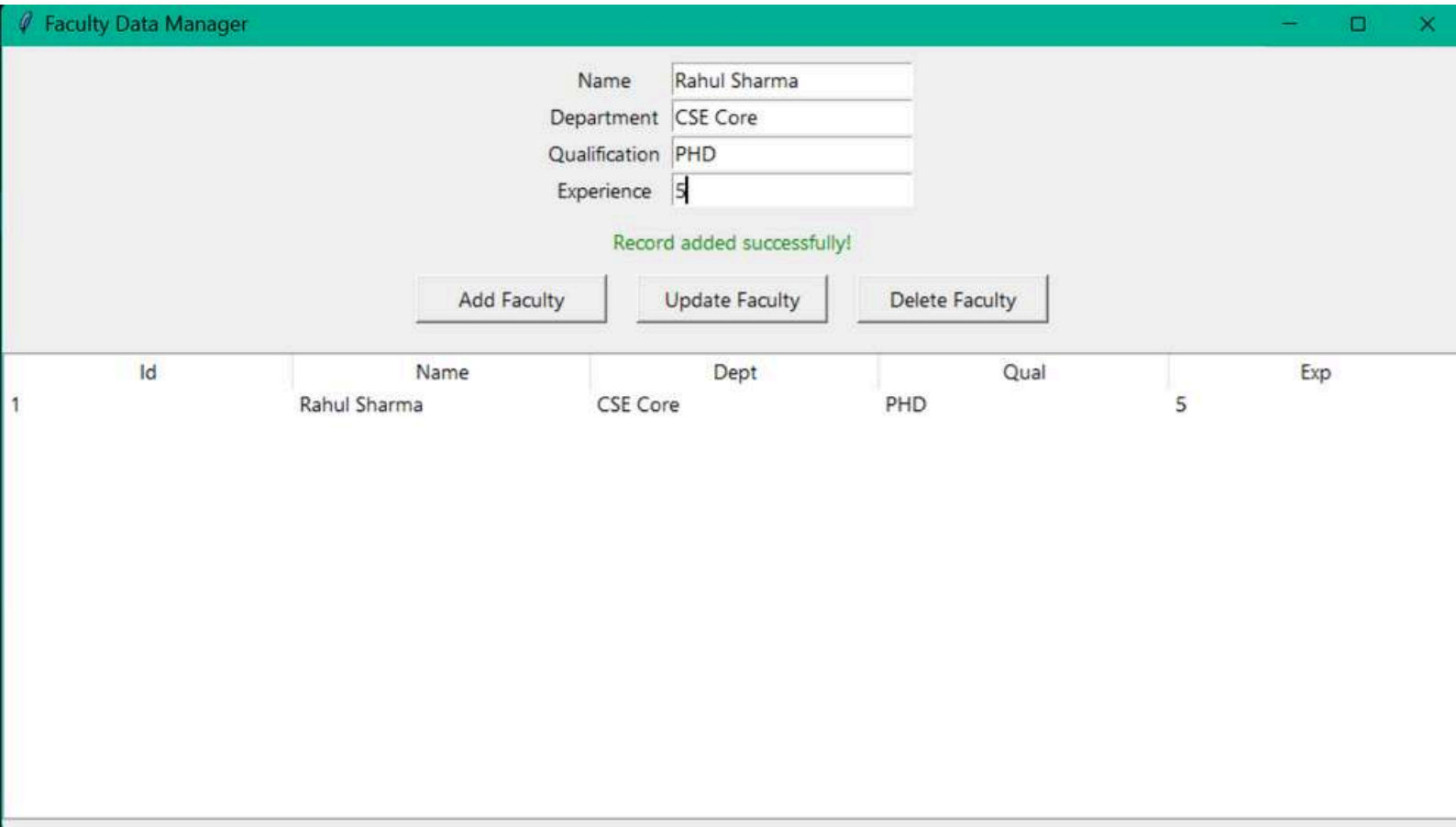
This is the SQLite database file that stores all faculty records permanently.

SCREENSHOTS

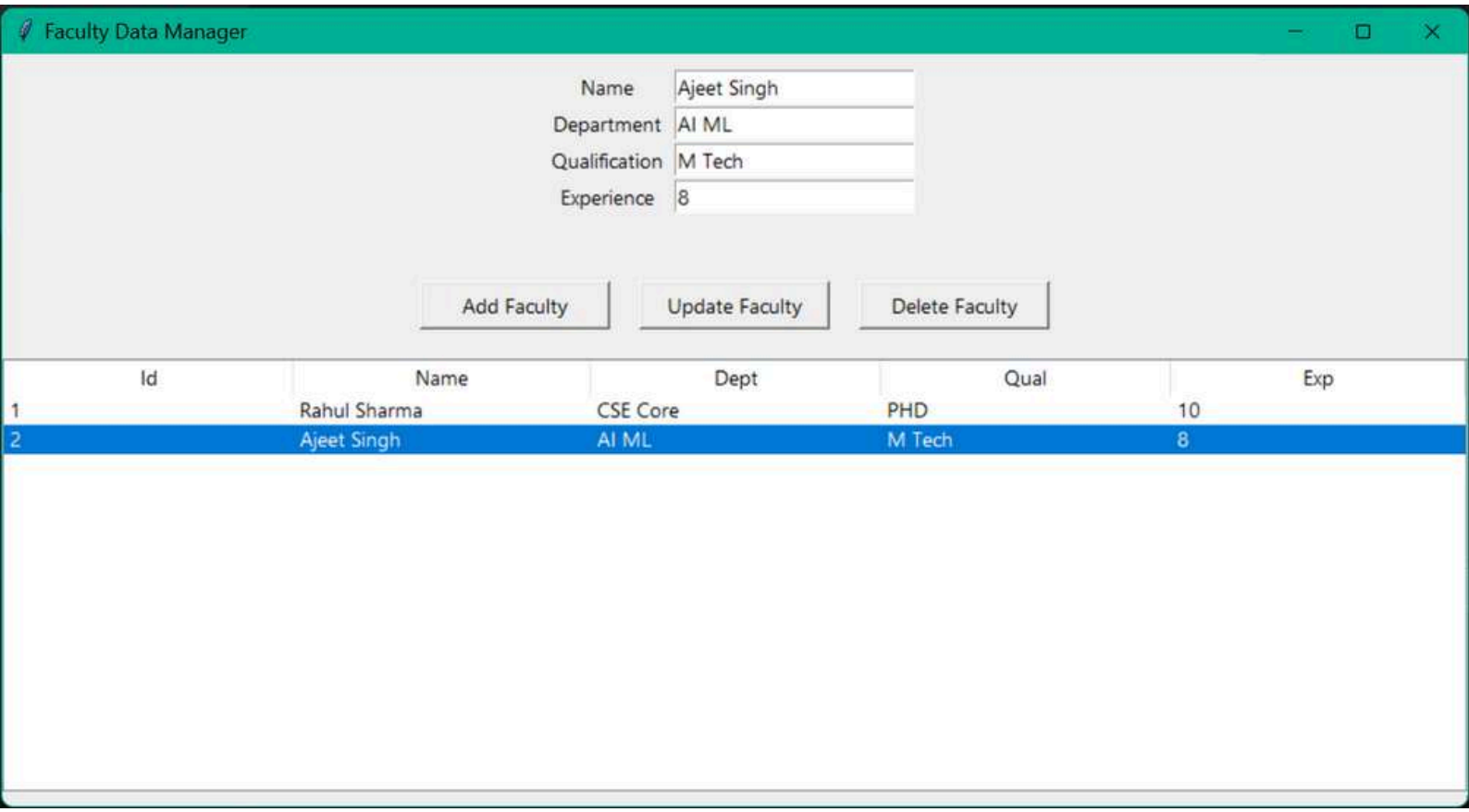
① Home Screen



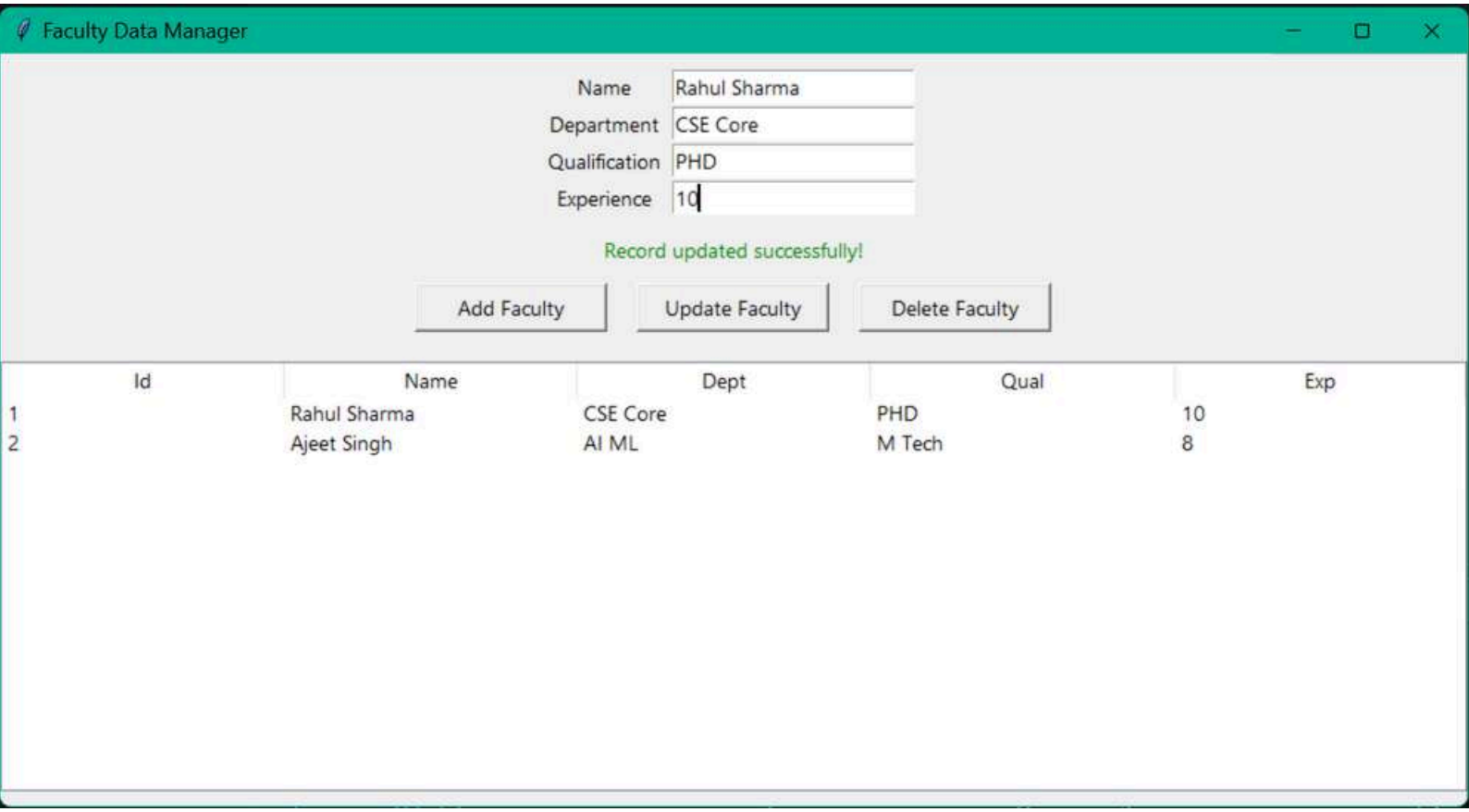
② Adding a Record



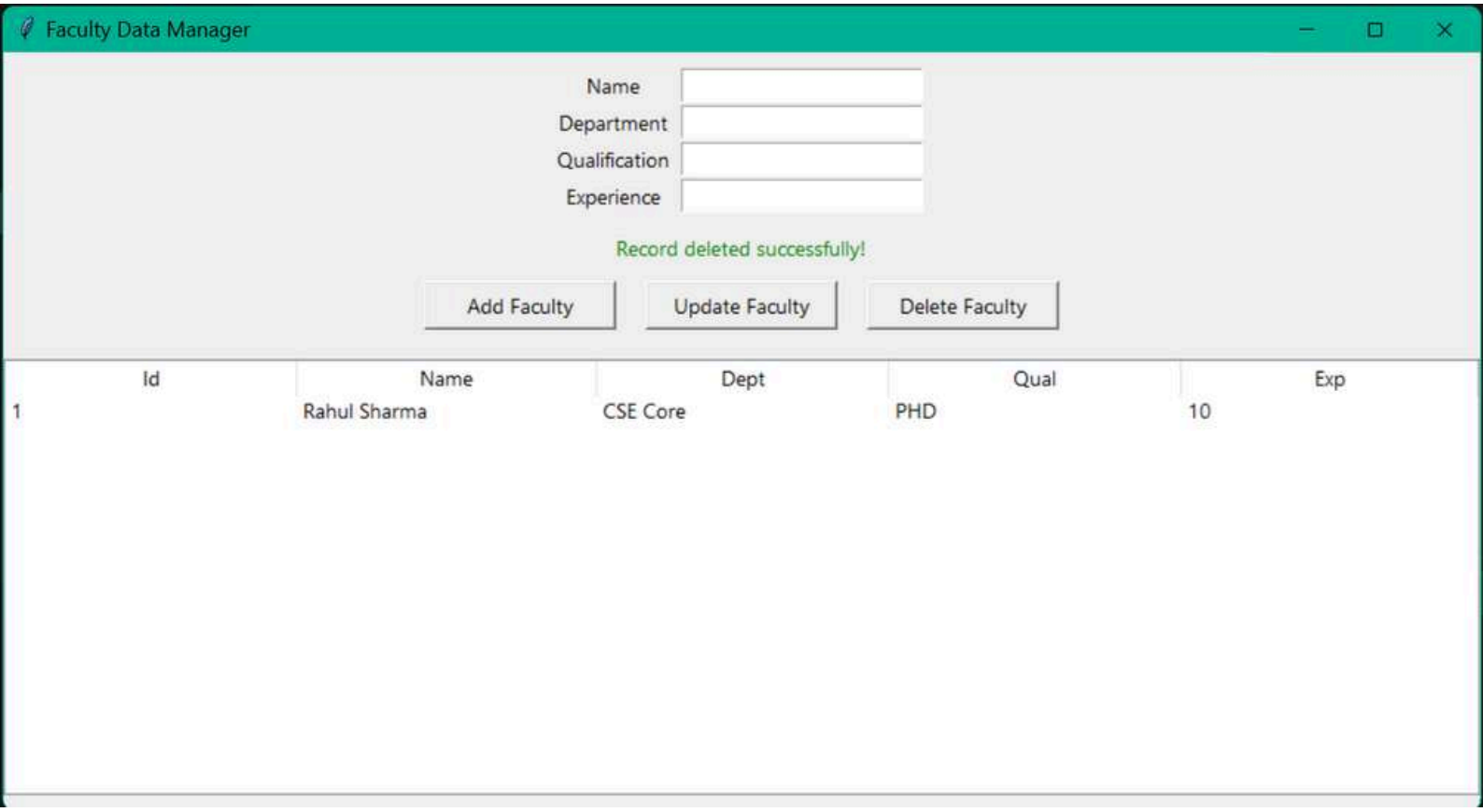
③ Selecting a record



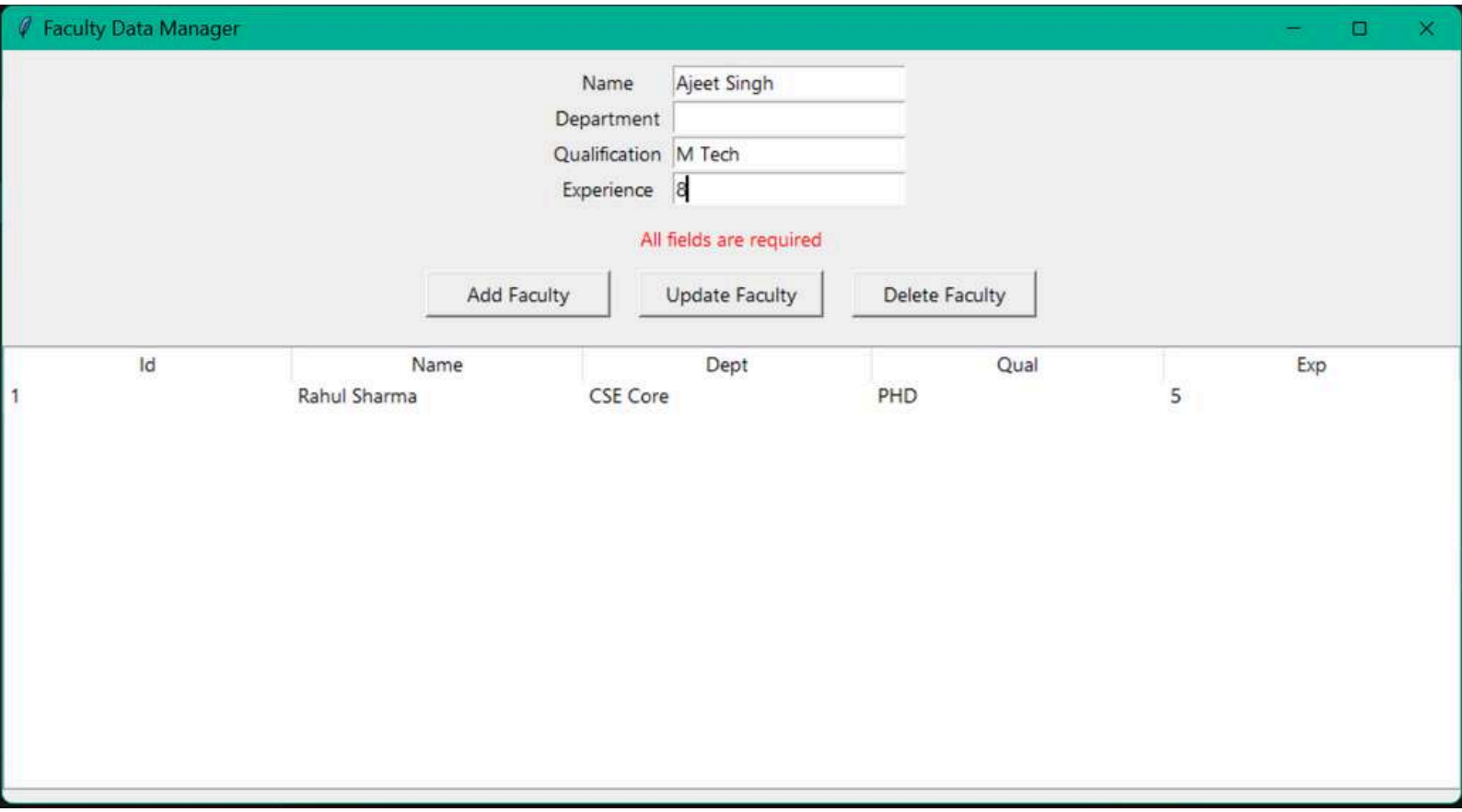
④ Updating a record



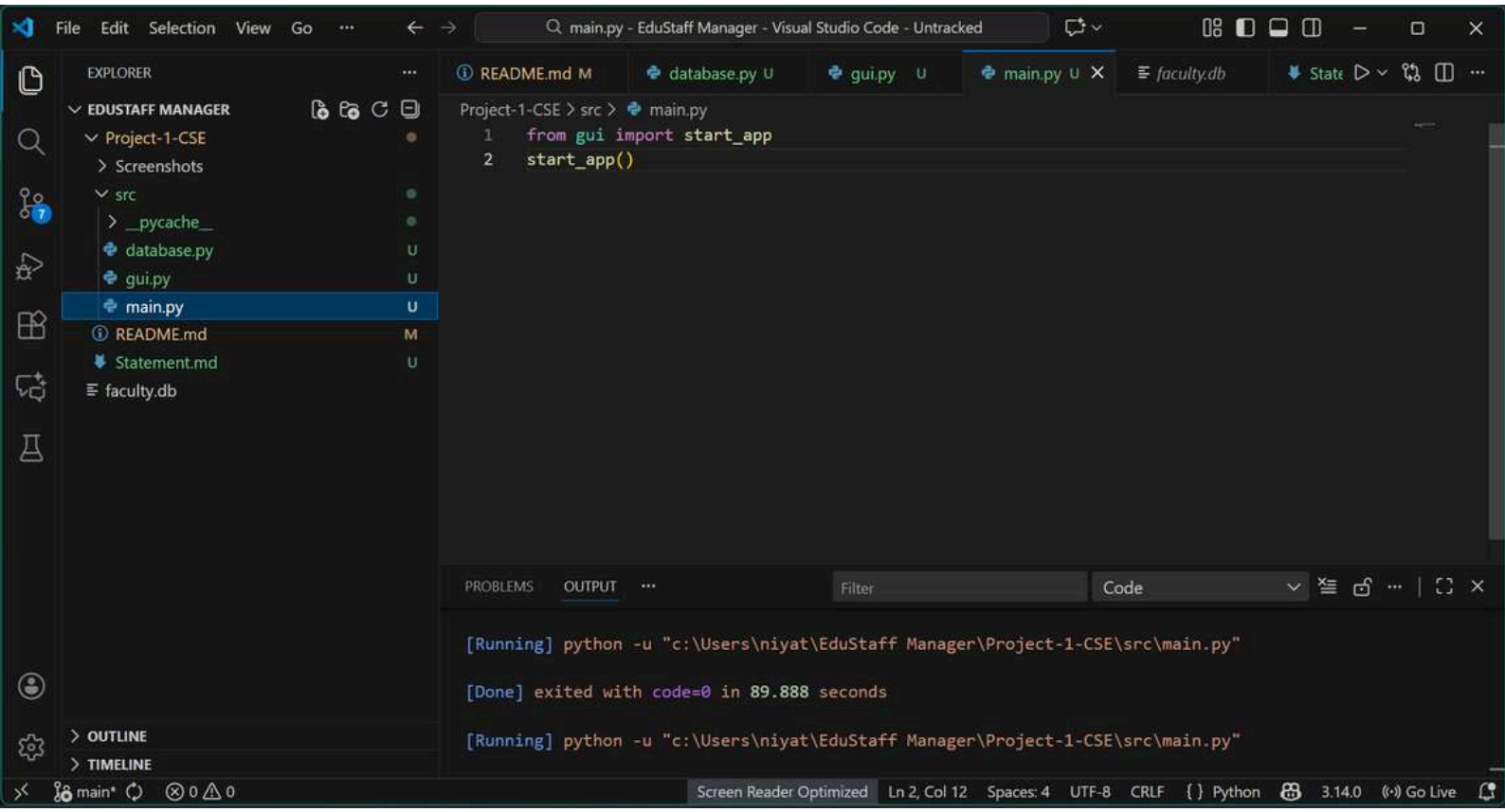
5 Deleting a record



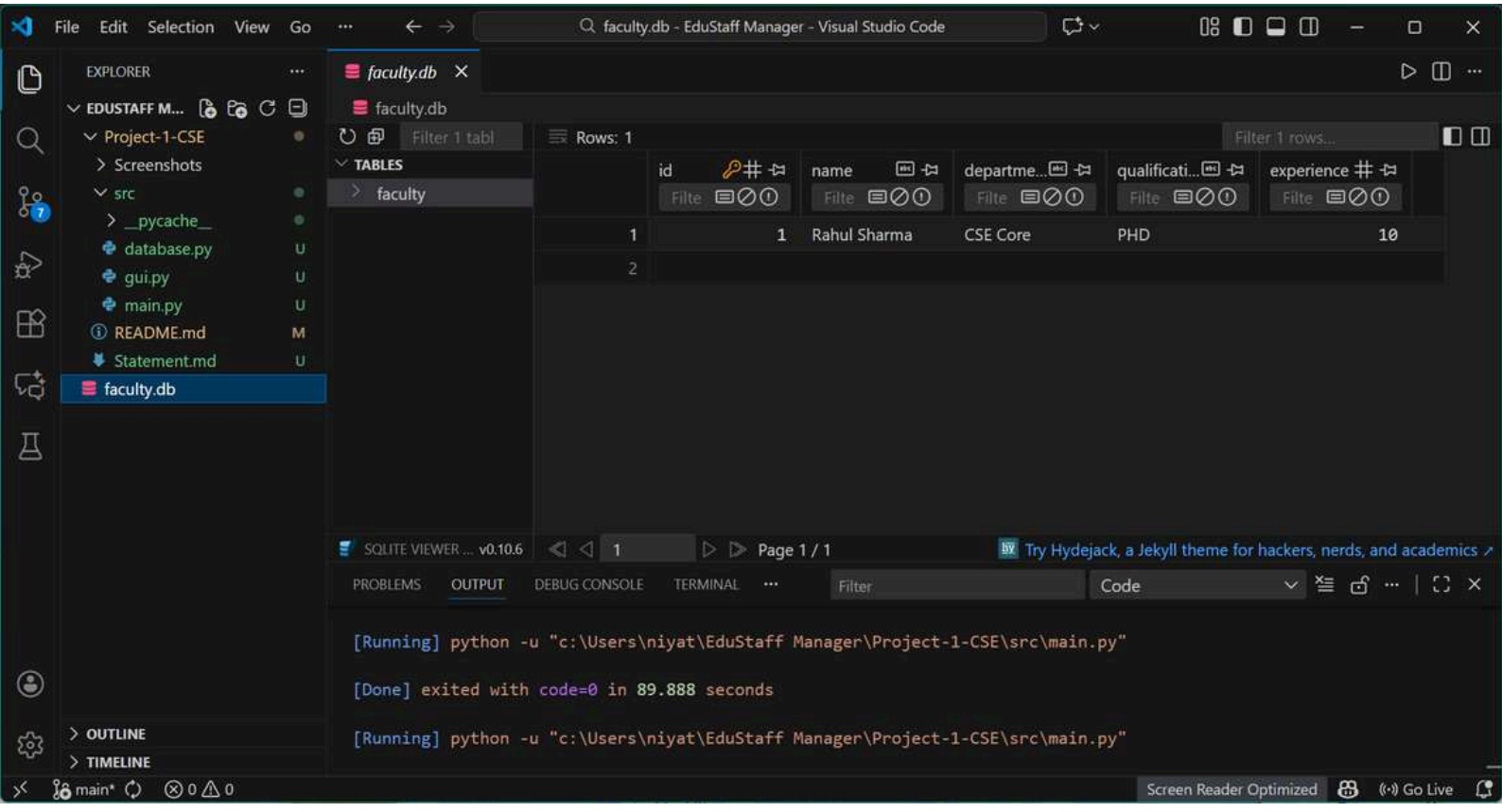
6 Validation Error



7 Project Structure



8 Database File



Testing Approach

Since this is a small application, I focused mainly on manual testing.
-Here are the tests I performed:

Add Operation

Entered valid details and checked if the record appears in the table.

Empty Input Testing

Tried adding a record with missing details to confirm the error message appears.

Update Operation

Selected a record, edited the fields, and checked if the table updated correctly.

Delete Operation

Selected a record and clicked delete to ensure it disappeared.

Database Verification

Opened the SQLite database to confirm that insert, update, and delete operations actually changed the stored data.

Multiple Entries

Added many records to ensure the table still scrolls and works smoothly.

The application performed correctly for all tests.

Challenges Faced

1. **Understanding how GUI and database connect**
   In the beginning, it was confusing to figure out how Tkinter buttons link to database functions. After practicing, it became clearer.

2. **Treeview Selection Handling**
   Getting the selected row's data and automatically filling it back into the text fields took time to understand, especially the part where Treeview stores values.

3. **Refresh Issues**
   Sometimes after adding or updating, the table didn't refresh properly. Fixing this required re-calling the load_records() function at the right places.

4. **Managing File Structure**
   Keeping separate files for GUI, database, and main program was new for me. Initially, I mixed things, but later understood how to structure the project neatly.

5. **Avoiding Invalid Inputs**
   At first, the app allowed empty values. Adding proper validation and message boxes improved the reliability.

Key Learnings

Working on this project taught me several important things, both technical and practical.

Overall, this project was a very good introduction to building a complete mini-application from scratch.

- Understanding CRUD operations
- Connecting different components
- Tkinter basics
- Debugging skills
- Project structuring
- Using SQLite database

Future Enhancements

Search functionality	A search box to quickly find a faculty member by name or department.
Sorting Options	Allow sorting the table by experience, department, alphabetical order, etc.
Export to Excel/CSV	A button to export all data into a file for easy sharing.
Login System	Add user login so only authorized users can modify the database.
Better UI Design	Improve the look of the GUI using custom Tkinter themes or a more modern library.
Multiple Tables	Add related features like student data or time-table management.

Conclusion

This project helped me put together everything I learned in the course into one working application. Building the Faculty Data CRUD Manager gave me practical experience in connecting a GUI with a database, handling user input, and organizing a real application properly.

Even though the project is small, it showed me how complete systems are built step by step—from designing the structure to implementing the code and finally testing it.

Overall, this project was a great learning experience and gave me confidence to take on bigger applications in the future.

REFERENCES

These are the resources I used to understand concepts and fix issues during development:

- Python Official Documentation (Tkinter & SQLite)
- StackOverflow discussions for Treeview and Tkinter-related errors
- GeeksforGeeks articles on Tkinter and SQLite basics
- YouTube tutorials on CRUD applications using Tkinter and SQLite
- Class notes and materials from the course