**“Student Management System using PL/SQL and Python Tkinter GUI”**

**A PROJECT REPORT**

***Submitted by***

**ANSH CHOPRA (24MCI10105)**

*in partial fulfillment for the award of the degree of*

**MASTER OF COMPUTER APPLICATION**

**IN**

**ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING**



**Chandigarh University**

OCTOBER 2024

**ACKNOWLEDGEMENT**

I would like to express my sincere gratitude to **Miss Shruti Sharma** , my project supervisor, for his invaluable guidance, support, and encouragement throughout the duration of this project. Her expertise and insights were instrumental in the successful completion of the project, "Student Managemnet System using PL/SQL and Python Tkinter GUI."

I am also thankful to **Chandigarh University** for providing the resources and a conducive environment that enabled me to explore and apply algorithmic principles in solving real-world problems. This project has been a significant learning experience, enhancing both my theoretical knowledge and practical skills.

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Topic** | **Page No.** |
| **1** | Introduction | 1 |
| **2** | **Software and Hardware Requirements** | 1-2 |
| **3** | **Functionality Overview** | 2 |
| **4** | **Technologies Used** | 3 |
| **5** | **System Features** | 3-4 |
| **6** | **System Design** | 4-5 |
| **7** | **Key Code Snippets** | 5-6 |
| **8** | **Testing and Validation** | 6-7 |
| **9** | **Challenges and Solutions**: | 7 |
| **10** | **Future Enhancements**: | 7 |
| **11** | **Conclusion** | 8 |
| **12** | **Project Output/Screenshot** | 8-9 |
| **13** | **References** | 9 |

****1. Introduction****

This report provides a comprehensive overview of the Enhanced Student Management System, an application that merges the flexibility and user-friendliness of Python's Tkinter GUI with the robustness of a PL/SQL (MySQL) backend database. This system is designed to handle basic student management tasks, including adding new records, viewing existing data, deleting records, and searching for specific entries based on name. The system is built with an emphasis on ease of use, security, and modular design, allowing for further expansions and customizations.

In an era where digital solutions are becoming increasingly essential for managing educational data, the Enhanced Student Management System offers an efficient, streamlined way for institutions and individuals to manage student records. It stands out with its comprehensive CRUD (Create, Read, Update, Delete) capabilities and user-centric design.

**2. Software and Hardware Requirements**

**Software Requirements:**

* **Programming Language**: Python 3.x is required due to its extensive library support and ease of integration with other systems.
* **Database Management System**: MySQL is used as the database backend for its reliability, scalability, and support for PL/SQL.
* **Library Dependencies**:
  + **mysql-connector-python**: Used for establishing communication between the Python application and the MySQL database.
  + **tkinter and ttk**: Utilized for building the graphical interface and interactive elements.
* **Development Environment**: The project can be developed in any modern Python IDE, such as PyCharm or VSCode, which provides tools for code linting, debugging, and version control.

**Hardware Requirements:**

* **Processor**: A minimum 2 GHz dual-core processor is recommended for smooth performance.
* **RAM**: At least 4 GB of RAM to handle the Python interpreter and database operations without lag.
* **Storage**: A minimum of 500 MB for database storage and an additional 200 MB for Python environment and related files.
* **Operating System**: Compatible with Windows, macOS, or Linux to ensure broad accessibility.

**3. Functionality Overview**

The Enhanced Student Management System has been designed with user-centric features, making it intuitive and easy to navigate. The core functionalities include:

* **Add Student**: Users can input student details, such as name, age, and major, into the system. Once validated, these details are stored in the database.
* **View All Students**: The system retrieves and displays all records in a structured, readable format.
* **Delete Student**: Users can delete a student record based on its unique ID.
* **Search Student**: Provides search functionality by name, allowing for partial or complete name matching.

These features collectively enable seamless interaction between the user and the database, making student record management simple and efficient.

****4. Technologies Used****

* ****Frontend**:**
  + **Python Tkinter Framework**: Provides an easy-to-use, flexible way to create a GUI that interacts with backend processes.
  + **Widgets Used**: ttk.Entry, ttk.Button, tk.Text are employed for input fields, buttons, and output display.
* ****Backend**:**
  + **PL/SQL (MySQL)**: Manages the data efficiently with features such as data validation, indexing, and secure transactions.
* ****Connector Library**:**
  + mysql-connector-python: Establishes a secure and seamless connection between the Python application and the MySQL database.

****5. System Features****

**User-Friendly Interface**: The GUI employs Tkinter's ttk widgets to ensure an aesthetically pleasing, intuitive interface. This allows users of all technical backgrounds to interact with the system effortlessly.

****Robust Database Operations**:**

* The system supports CRUD functionalities for comprehensive student record management.
* Operations include adding new records, viewing existing records, updating records, and deleting entries.

****Input Validation and Error Handling**:**

* The application ensures that fields such as age and ID are numeric, reducing errors during data entry.
* The system uses messagebox alerts to notify users of successful operations or to prompt them with error messages if input is invalid or incomplete.

****Responsive Layout**:**

* The application layout is designed to be responsive, allowing it to adjust seamlessly to different screen sizes while maintaining usability.

****6. System Design****

**Database Schema**: The database student\_db consists of a single table students that holds all student records. The table's structure ensures that each student entry is unique, with the id column set as the primary key and auto-incremented to simplify record addition.

**SQL**

CREATE TABLE students (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(255) NOT NULL,

age INT NOT NULL,

major VARCHAR(255) NOT NULL

);

**Python Functions**:

* connect\_db(): Establishes a connection to the MySQL database using mysql-connector-python.
* add\_student(): Validates input and inserts a new student record into the database.
* view\_students(): Retrieves and formats all records for display in the GUI.
* delete\_student(): Accepts a unique student ID, validates it, and deletes the corresponding record.
* search\_student(): Performs a search based on the provided name input, using partial matching for greater flexibility.
* clear\_output(): Clears the current output area for new data or improved readability.

**System Architecture**: The system architecture follows a client-server model, where the Python GUI serves as the client and MySQL functions as the server. The application logic sits within the Python client, which interacts with the database to perform operations.

**Workflow**:

1. **User Input**: Users interact with the GUI to input data or request information.
2. **Validation**: Input data is checked for correctness (e.g., age must be numeric).
3. **Database Interaction**: Upon validation, the application connects to the database to perform the required operation (insert, retrieve, delete, search).
4. **Output**: Results are displayed in the tk.Text widget, and message boxes inform the user of operation outcomes.
5. ****Key Code Snippets**:**

Below are some code snippets to illustrate the primary functions:

****Adding a Student**:**

def add\_student():

name = entry\_name.get()

age = entry\_age.get()

major = entry\_major.get()

if name and age.isdigit() and major:

conn = connect\_db()

cursor = conn.cursor()

query = "INSERT INTO students (name, age, major) VALUES (%s, %s, %s)"

cursor.execute(query, (name, int(age), major))

conn.commit()

cursor.close()

conn.close()

messagebox.showinfo("Success", "Student added successfully!")

entry\_name.delete(0, tk.END)

entry\_age.delete(0, tk.END)

entry\_major.delete(0, tk.END)

else:

messagebox.showwarning("Input Error", "Please provide valid input.")

****Viewing All Students**:**

def view\_students():

conn = connect\_db()

cursor = conn.cursor()

cursor.execute("SELECT \* FROM students")

records = cursor.fetchall()

cursor.close()

conn.close()

output\_text.delete("1.0", tk.END)

if records:

for record in records:

output\_text.insert(tk.END, f"ID: {record[0]}, Name: {record[1]}, Age: {record[2]}, Major: {record[3]}\n")

else:

output\_text.insert(tk.END, "No records found.")

1. ****Testing and Validation**:**

The system underwent rigorous testing to ensure functionality across various scenarios:

* **Add Student**: Validated inputs and confirmed successful addition to the database.
* **View All Students**: Checked that all records displayed correctly without errors.
* **Delete Student**: Verified deletion functionality with both valid and invalid IDs.
* **Search by Name**: Tested partial and exact name matches to confirm search accuracy.
* **Error Handling**: Simulated erroneous input to trigger warning messages and ensure system resilience.

****9. Challenges and Solutions**:**

* **Challenge**: Secure database connectivity and user data validation.
  + **Solution**: Used parameterized queries to safeguard against SQL injection and validated inputs thoroughly before processing.
* **Challenge**: Managing empty or incorrect inputs gracefully.
  + **Solution**: Integrated comprehensive checks and messagebox prompts to guide user actions and handle errors.

****10 Future Enhancements**:**

To further enhance the Enhanced Student Management System, the following improvements are planned:

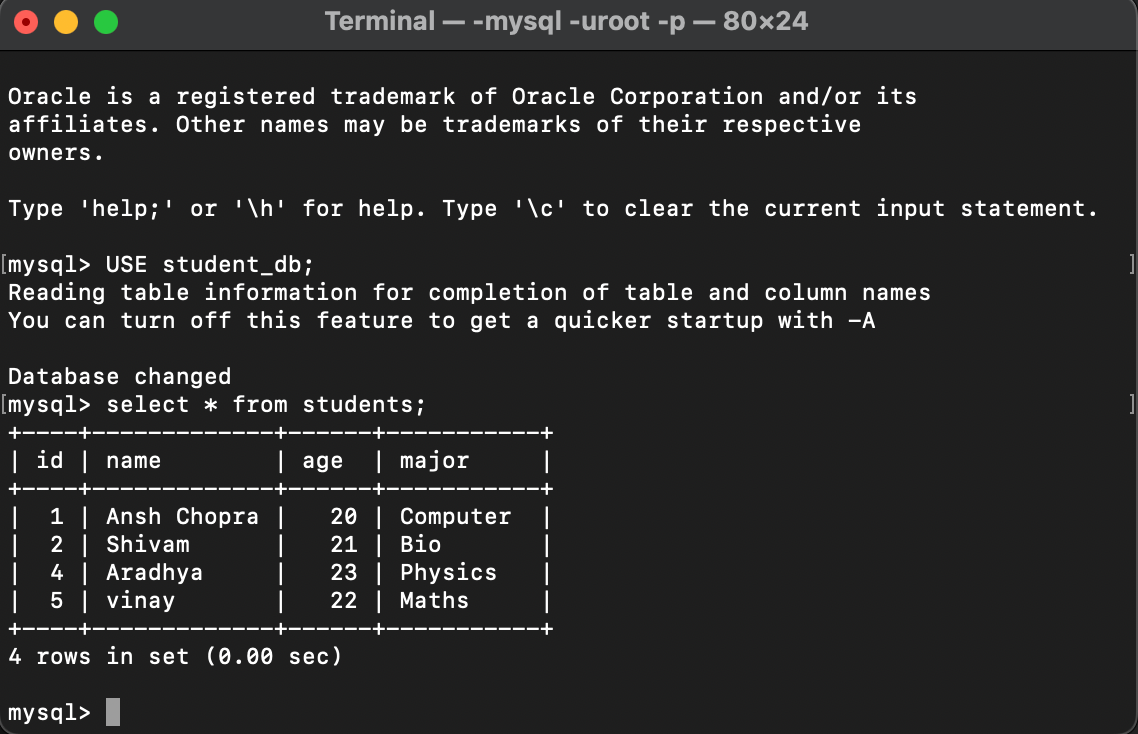
* **Additional Fields**: Include GPA and enrollment date to extend student record details.
* **Data Export**: Implement data export options (e.g., CSV or Excel formats).
* **User Authentication**: Introduce user roles and password protection for data security.
* **Course Enrollment**: Expand to manage student course registration and grades.

****11. Conclusion****

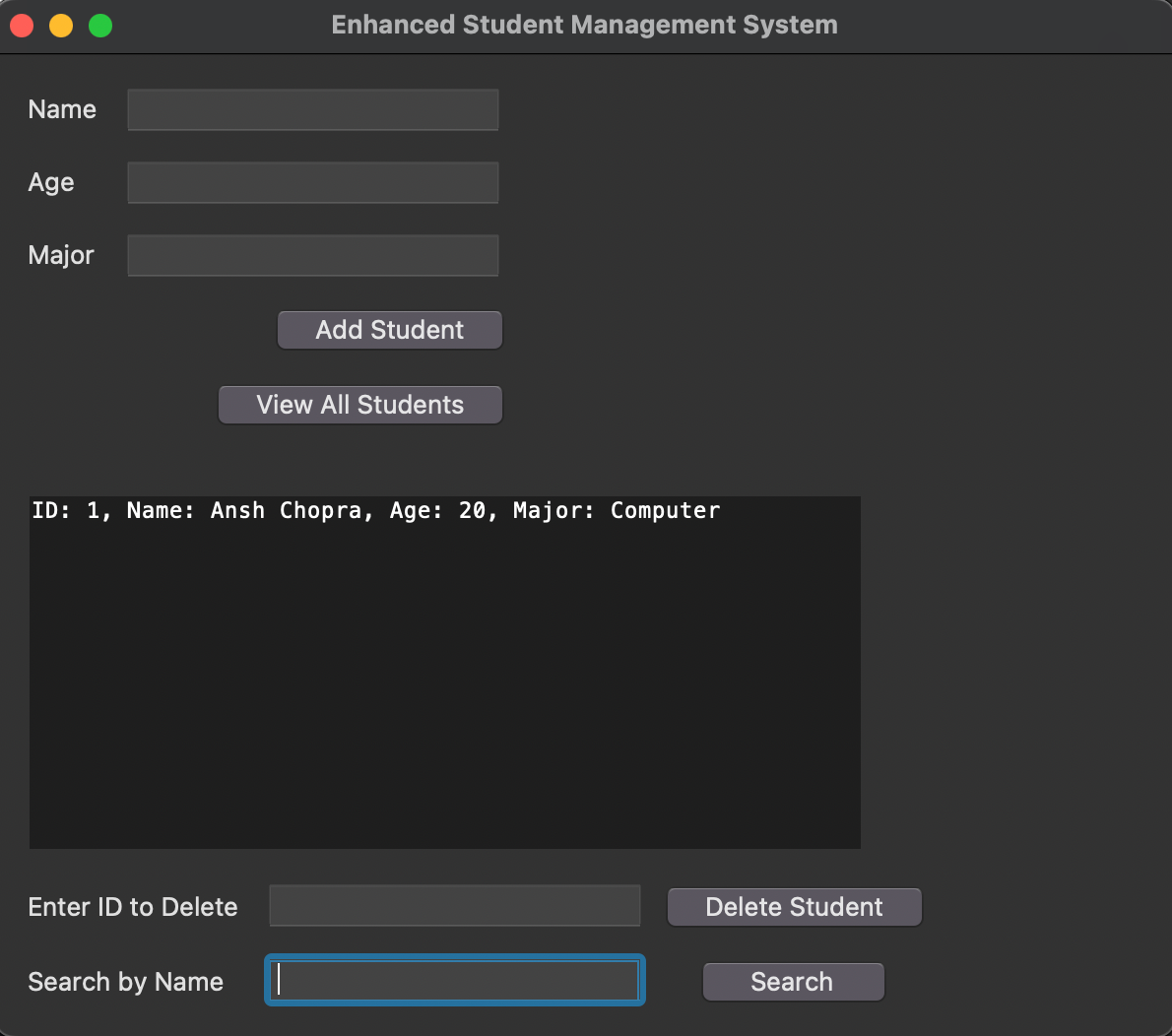
The Enhanced Student Management System successfully showcases the integration of Python and MySQL for efficient student record management. The system’s design prioritizes usability and scalability, positioning it as an essential tool for educational institutions aimieffective ng for data handling.

1. **Project Output/Screenshot**

**Backend**



**Frontend**



1. **References**

* [Python Tkinter Documentation](https://docs.python.org/3/library/tkinter.html" \t "/Users/ansh1011/Desktop/daa_project/x/_new): Comprehensive guide and reference for using the Tkinter library to build Python GUI applications.
* [MySQL Connector/Python Documentation](https://dev.mysql.com/doc/connector-python/en/" \t "/Users/ansh1011/Desktop/daa_project/x/_new): Official documentation for using MySQL Connector/Python to establish a connection between Python applications and MySQL databases.
* [PL/SQL Guide](https://www.oracle.com/database/technologies/appdev/plsql.html" \t "/Users/ansh1011/Desktop/daa_project/x/_new): A guide by Oracle providing detailed information about PL/SQL for effective database management and programming.