Final Project Title: Student Information Management System

NAME : MD AKASH HOSSAIN
ID : 2021521460115
Course Name : Database Systems
Faculty : School of Software Engineering

Abstract

An organized and systematic one solution is essential for all the institution and organization. There are various department of administration for maintain of the University information and student database in any institution. All these departments provides various records regarding students. Most of these track records needs to maintained information about students. This information could be the general details like student name, student id, major,Gender,enrollment year etc or specific information related to the departments like collection of data. This entire module in university is independent. These are maintained manually. So we need to automate and centralized as information from one module will be related to our modules. With that in mind we over halted the existing student information management system and made necessary improvement to streamline the process. Our work is useful for the easy user interface. We are planning to make the student information management system that can be used by the educational institutes to maintain the records of student easily. Archiving this objective is difficult of using the manual system as the information is scattered, can be redundant and collecting relevant information may be very time consuming. All these problems are solved using this project.

Keywords

Student Information Management System,Database Design,Data operations,Data analysis and visualization,Testing

Introduction

The Student Information Management System (SIMS) is an essential tool for modern educational institutions, designed to maintain comprehensive records of both college and university students. In today's digital age, the transition from manual to computerized systems has revolutionized the way administrative tasks are managed, bringing efficiency and accuracy to various processes.The primary function of the SIMS is to handle all activities that were traditionally performed manually, now efficiently managed through computerized methods. These activities encompass a broad range of functions including student online registration, update student,all the course processes, staff and teacher management, course management, and course book management.Another crucial aspect of SIMS is the maintenance of comprehensive student records. Detailed information including background information, educational qualifications, personal details, and resumes are stored within the system. This centralized storage of student data ensures that information is easily accessible for both students and university administrators. It facilitates the management of student records and simplifies the process of data retrieval.The system's **search and query functionality** is particularly valuable for administrators. For example, if there are 200 students and a specific student's information is needed, this system streamlines the process. Instead of sifting through numerous records manually, administrators can simply enter the student's name into the search criteria, and the system will quickly retrieve the relevant information, including the student's ID and other pertinent details.

Additional Features

- **Create Student:**Administrators can add new student records to the system, capturing essential details such as name, enrollment year, major, and personal information. This ensures that all new students are properly registered and their data is accurately recorded. The streamlined process facilitates quick and efficient onboarding of new students, allowing them to commence their academic journey without delay.

- **Delete Student:**The system allows for the removal of student records that are no longer needed, ensuring the database remains up-to-date and relevant. This might include students who have graduated, transferred to another institution, or withdrawn from the program. By keeping the database current, the institution ensures that only relevant and active records are maintained, which simplifies data management and improves the overall efficiency of the system.
- **Update Student**:Student information can be updated as needed, ensuring that all records reflect the most current and accurate data. This capability is vital for maintaining the integrity of the student database, as it accommodates changes such as updated contact information, changes in major, or corrections to previously entered data.
- **Create Book Search Option:**A book search feature enables students and administrators to search for course-related books, streamlining the process of finding and purchasing textbooks. By providing a centralized platform for book searches, the system ensures that students can easily access the necessary learning materials for their courses.
- **Create Course**:New courses can be added to the system, complete with details such as course ID, name, description, and schedule. This ensures that all new courses are properly documented and available for student registration. By efficiently managing course creation, the institution can keep its curriculum offerings up-to-date and relevant to the needs of its students.
- **Assign Book to Course:**Administrators can link specific books to particular courses, ensuring students have access to the necessary materials for their studies. This feature not only facilitates the smooth running of courses but also ensures that students are adequately prepared with the necessary resources, thereby enhancing their learning experience.
- **Teacher Course Management:**The system allows for the assignment of teachers to specific courses, managing their schedules and ensuring that all classes are adequately staffed. This functionality helps in balancing teacher workloads and ensures that students receive instruction from qualified educators in a timely and organized manner.
- **Teacher Information Management:**Detailed records of teacher information are maintained, including their qualifications, courses taught, and schedules. This ensures that the institution has accurate and up-to-date information on its faculty, which is essential for planning, accreditation, and overall academic administration. By managing teacher information effectively, the institution can ensure high standards of teaching and academic excellence.

The benefits of the Student Information Management System are manifold. Firstly, it significantly enhances efficiency by automating routine tasks and reducing the workload on administrative staff. Secondly, it improves the accuracy of student records by minimizing human errors associated with manual data entry. Thirdly, it offers convenience for students, who can access and manage their information from any location. Lastly, it ensures better data security and privacy by restricting access to authorized users only.

Problem Statement

The current system of any university and college is manual paper based system. This manual system is very difficult to maintain the information of student. It is a very tedious system and time consuming. To overcome all these problems we have to decide to make the computerized student information management system which help to manage the student record electronically to replace the paper based system.

Objective of this project

The objective of the student information management system is to allow the administrator or any organization to edit and find the personal details of student and allow the student to keep up to date his profiles. It also facilities the keeping all records of student Name, Student id, course and course relate books,course teacher etc. So all the information about an student will be available in few seconds. Overall it'll make the student information management an easier job for administrator and the student of any organization. The main of SIMS document is to illustrate the illustrate the requirement of the project student information system and intended to help any organization to maintained and manage the student personal data.

Methodology :Database Connection and Setup :

The initial step in the methodology involves establishing a connection to the SQLite database named university.db. This is achieved through the sqlite3.connect method, which is a standard approach for interfacing with SQLite databases in Python. Upon establishing the connection, a cursor object is created using the connection.cursor method. The cursor acts as an intermediary, allowing for the execution of SQL queries and the manipulation of the database.

Creating Tables

The next phase involves the creation of several key tables necessary for organizing the university's data. Each table is designed to store specific types of information pertinent to the university's operations.

❖ **Creating the Students Table**

The first step in managing student data is to create a dedicated table in the database specifically for storing student information. This table, typically named Students, is designed to include several key attributes that uniquely identify and describe each student.**StudentID:** A unique identifier assigned to each student. This serves as the primary key for the table, ensuring that each student can be uniquely identified.

Student Name: The full name of the student. This attribute is essential for identifying and referring to the student.**Major:** The field of study or major that the student is pursuing. This helps in categorizing students based on their academic focus.**Gender:** The gender of the student. This information can be useful for demographic analysis and reporting.

❖ **Courses Table:**

The Courses table is created to store details about the various courses offered by the university. It contains columns for CourseID, CourseName, and TeacherID. The TeacherID column is a foreign key that references the TeacherID in the Teachers table, establishing a relationship between courses and their instructors.

❖ **Books Table:**

The Books table is designed to catalog books relevant to the courses. It includes BookID, Title, and Major. The BookID is the primary key, ensuring each book can be uniquely identified.

❖ **CourseBooks Table:**

This table serves to link courses with the books that are required or recommended for them. It contains CourseID and BookID columns, both of which are foreign keys referencing the Courses and Books tables, respectively. This many-to-many relationship table allows for a flexible association between courses and books.

❖ **TeacherCourse Table:**

Similar to the CourseBooks table, the TeacherCourse table is established to link teachers with the courses they teach. It includes CourseID and TeacherID, both serving as foreign keys. This setup allows for the documentation of which teachers are responsible for which courses.

Inserting Data

With the tables created, the next step involves populating them with data. This step ensures that the database is not just a structure but also contains meaningful information.

✧ **Inserting Student Data:**

Inserting data for student records is a critical process in database management, ensuring comprehensive and organized information for each student. This involves collecting essential details such as StudentID, Name, Major, Gender. Accuracy is paramount; thus, data verification and consistency checks are necessary to avoid errors. Each StudentID must be unique, serving as a primary key to prevent duplicate entries. Data can be entered manually for small datasets or imported automatically for larger ones, using tools to streamline the process. Ensuring fields like Name and Major are not null maintains data integrity, while consistent formatting supports reliable record-keeping. This meticulous approach ensures a robust and functional database for managing student information efficiently.

✧ **Inserting Teachers Data:**

Multiple records are inserted into the Teachers table. Each record includes a unique TeacherID, along with the teacher’s name, department, and gender. This data is crucial for identifying and organizing the university's faculty.

✧ **Inserting Courses Data:**

Similarly, the Courses table is populated with records. Each course is given a unique CourseID, a descriptive CourseName, and a TeacherID that links it to a specific teacher. This setup provides a clear association between courses and their instructors.

Inserting Books Data:

The Books table is filled with records that include a unique BookID, the book's title, and the major or field of study it pertains to. This information is essential for students and faculty to identify relevant reading materials.

✧ **Inserting CourseBooks Data:**

Records are inserted into the CourseBooks table to establish the relationship between courses and books. Each record links a CourseID to a BookID, indicating which books are associated with which courses.

✧ **Inserting TeacherCourse Data:**

The TeacherCourse table is populated with records that link CourseID and TeacherID, documenting the teaching assignments of faculty members.

✧ **Committing Transactions**

After inserting the data, the connection.commit method is called. This step is critical as it saves all changes made to the database, ensuring that the newly created tables and inserted data are permanently recorded. Without committing these transactions, the changes would be lost once the connection to the database is closed.

✧ **Closing the Connection**

The final step in the methodology is to close the connection to the database using the connection.close method. Closing the connection is a good practice as it frees up resources and ensures that the connection to the database is not left open unnecessarily. This step marks the completion of the database setup and data insertion process.

System Design for Student Information Management System (SIMS)

Scope and Purpose

The primary purpose of designing a SIMS is to centralize and automate the management of student-related information across the institution. It aims to provide a cohesive platform that supports various functions such as student enrollment, attendance tracking, academic performance monitoring, and communication with stakeholders. The scope includes defining clear objectives for data accuracy, accessibility, security, and usability to meet the needs of diverse users within the educational ecosystem.

Overall System Design Objectives

A Student Information Management System (SIMS) is designed with key objectives in mind: efficiency through automation of tasks like enrollment and grading; robust data integrity and security measures including encryption and access controls to protect sensitive student information; and a user-centric interface that is intuitive and accessible across devices, catering to students, teachers, and administrators. By automating routine tasks, SIMS reduces manual effort and minimizes errors, enhancing operational efficiency in educational institutions. Data security measures ensure that student information remains confidential and protected from unauthorized access or loss, supported by regular data backups. The user-centric interface improves usability, allowing seamless access to academic records, communication tools, and administrative functions, ultimately enhancing the overall educational experience.

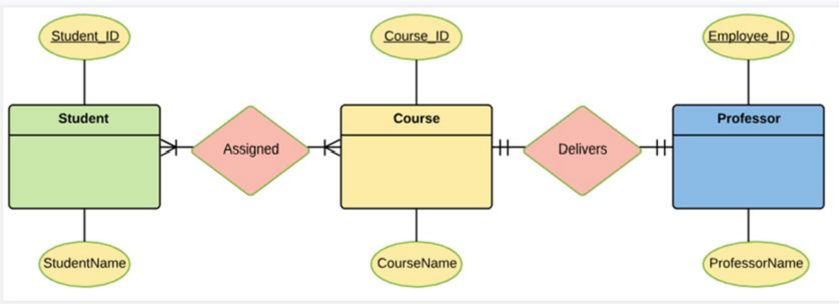
Database Schema Overview

The database schema comprises several core entities:

- **Teachers:** Stores detailed information about each teacher, including their unique identifier (TeacherID), name, department (Dept), and gender. This table forms the basis for managing teaching assignments and faculty information.
- **Courses:** Contains essential details about each course offered by the institution, such as CourseID, course name (CourseName), and the TeacherID of the instructor assigned to teach the course. This table enables efficient course scheduling and management.
- **Books:** Stores information about books relevant to different subjects or majors, identified by a unique BookID and including attributes like title (Title) and associated major (Major). This table supports the curriculum by linking relevant reading materials to courses.
- **CourseBooks:** Facilitates the many-to-many relationship between courses and books, indicating which books are assigned as required reading for each course. This setup ensures that course curricula are adequately supported by appropriate resources.
- **TeacherCourse:** Establishes another many-to-many relationship, linking teachers (TeacherID) to the courses (CourseID) they are assigned to teach. This table ensures accurate assignment of teaching responsibilities and facilitates workload distribution among faculty members.

Entity Relationships and Constraints

- **Teachers and Courses Relationship:** Each teacher can teach multiple courses, which is managed through the TeacherCourse table. This relationship ensures that teachers are appropriately assigned to courses based on their expertise and availability.
- **Courses and Books Relationship:** Courses may require multiple books as reading materials, captured in the CourseBooks table. This allows for flexibility in curriculum design and supports varied educational needs across different subjects.
- **Data Integrity and Constraints:** The database maintains data integrity through primary key constraints (TeacherID, CourseID, BookID) and foreign key relationships. These constraints ensure that each record is uniquely identifiable and that referential integrity is maintained across related tables.



Data operations and Implementation

Implementing a Student Information Management System (SIMS) involves translating technical specifications into a functional computer system through programming and development. This process ensures the system effectively handles tasks such as enrollment, grading, and reporting, tailored to the specific needs of educational institutions. Multiple implementations may exist for SIMS, reflecting varying specifications and standards. The implementation process consists of two main activities: programming software components to align with design specifications and deploying the system into operational mode within the institution's infrastructure. Key considerations include maintaining data integrity and security through encryption and access controls, alongside creating user-friendly interfaces for seamless interaction among students, teachers, and administrators. Successful implementation enhances operational efficiency, streamlines administrative tasks, and improves overall educational management and decision-making processes.

CRUD Operations Implementation

Create, Read, Update, Delete (CRUD) operations are pivotal for managing the SIMS database effectively:

- **Create (Insert):** Implemented through functions like create_student, create_course, create_book, assign_book_to_course, and assign_teacher_to_course, these operations utilize SQL INSERT statements to ensure atomicity and consistency when adding new student records, courses, books, and their relationships.
- **Read (Query):** Utilizing functions such as query_student, courses_with_teachers, bookCountByCourse, and teachersWithManyCourses, these operations execute optimized SQL SELECT queries tailored to retrieve specific information such as student details, course assignments with respective teachers, book counts per course, and teacher workload distribution.
- **Update:** Ensured by functions like update_student, update_teacher, and update_course, which employ SQL UPDATE statements to modify existing records accurately. These functions maintain data integrity by validating input and executing transactions within the SQLite environment.
- **Delete:** Managed through functions such as delete_student, delete_teacher, and delete_course, these operations execute SQL DELETE statements to remove obsolete records from the database, ensuring data hygiene and compliance with institutional data management policies.

Data analysis and visualization

In a Student Information Management System (SIMS), effective data analysis is crucial for understanding student demographics, optimizing teacher workload, managing courses, and planning resources. The provided Python script exemplifies this by leveraging SQLite, a lightweight relational database, to execute SQL queries and derive valuable insights from the database (student_info_system.db).

■ Student Information Analysis:

The script begins by analyzing student demographics and enrollment trends through several key functions: studentNumber() counts and prints the total number of students enrolled, providing a snapshot of the student population.studentCountByYear() retrieves and displays the count of students enrolled each year, grouped by EnrollmentYear. This function highlights enrollment trends over time, aiding in forecasting and resource allocation based on historical data.mostCommonStudentMajor() identifies the most popular major among students by counting occurrences and ordering them in descending order. This insight helps educational institutions tailor their academic offerings and allocate resources according to student interests and demand.

■ Teacher and Course Information Analysis:

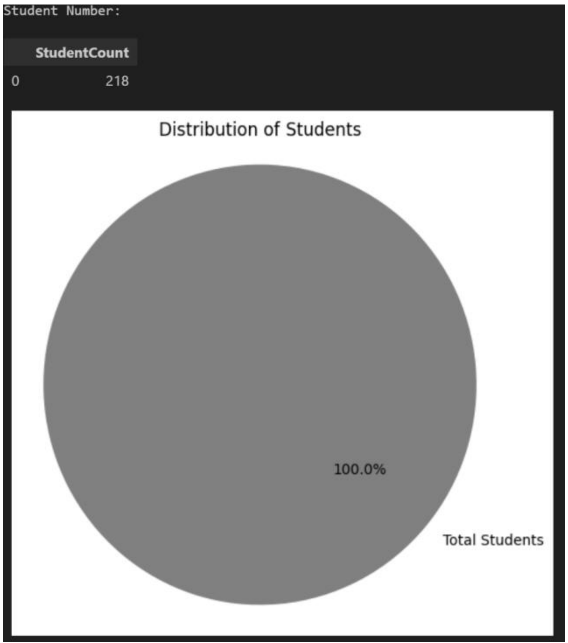
Next, the script delves into analyzing teacher workload and course distribution:courseCountByTeacher() lists the number of courses each teacher is associated with, providing a comprehensive view of teacher workload. This information is essential for balancing teaching responsibilities and optimizing teaching assignments. coursesWithTeachers() retrieves and prints courses along with their assigned teachers, facilitating effective course management and ensuring transparency in course assignments.teachersWithManyCourses(minCourses) identifies teachers handling more than a specified number (minCourses) of courses. This analysis helps in identifying high workload scenarios and redistributing courses for better workload management and instructional effectiveness.

■ Course and Book Details Analysis:

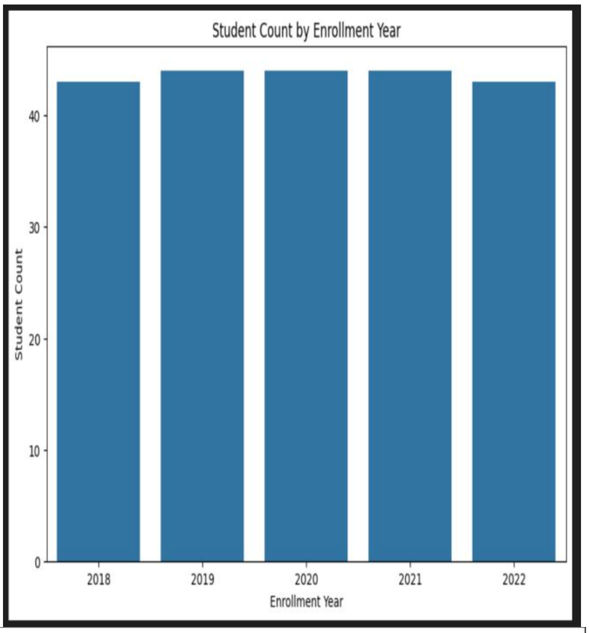
The script also provides insights into course-specific details:
bookCountByCourse() counts and displays the number of books associated with each course. This analysis supports resource planning and curriculum development by identifying course material requirements and ensuring adequate resources are available for each course.

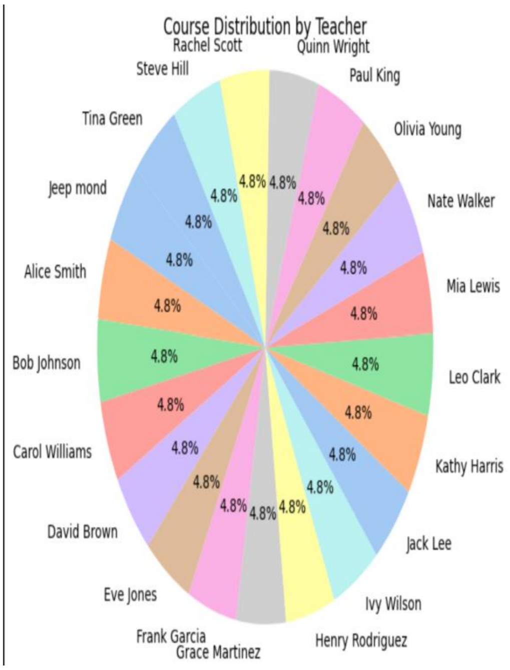
■ Teacher Workload Analysis:

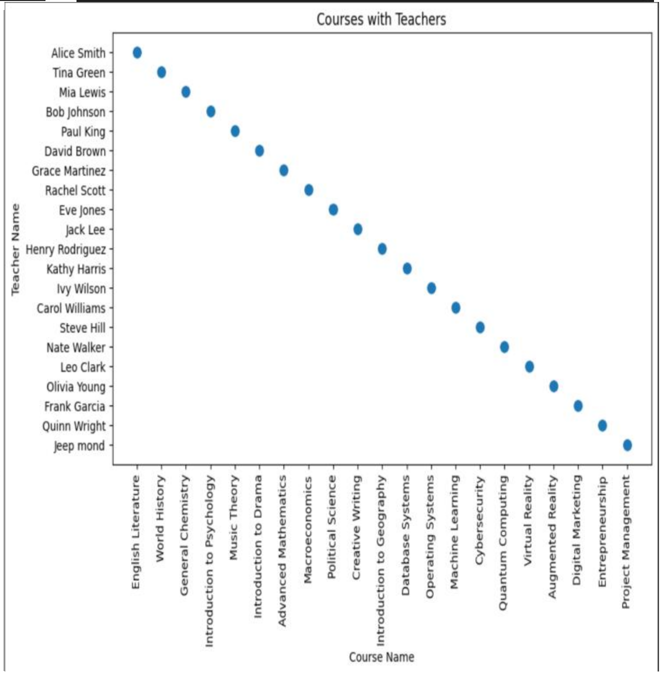
averageCoursesPerTeacher() calculates the average number of courses each teacher is responsible for. This metric aids in workload management and staffing decisions, ensuring equitable distribution members

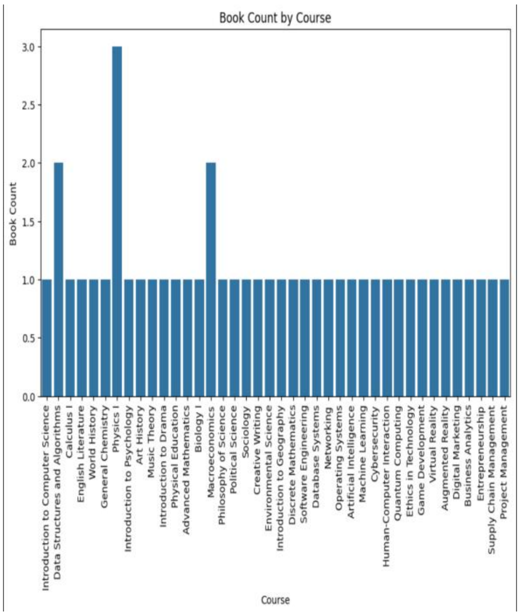


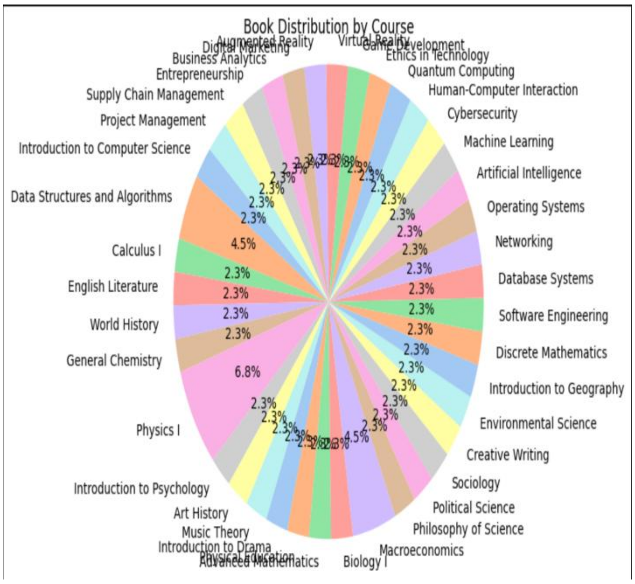
```
Student Number:

   StudentCount
0           218
```

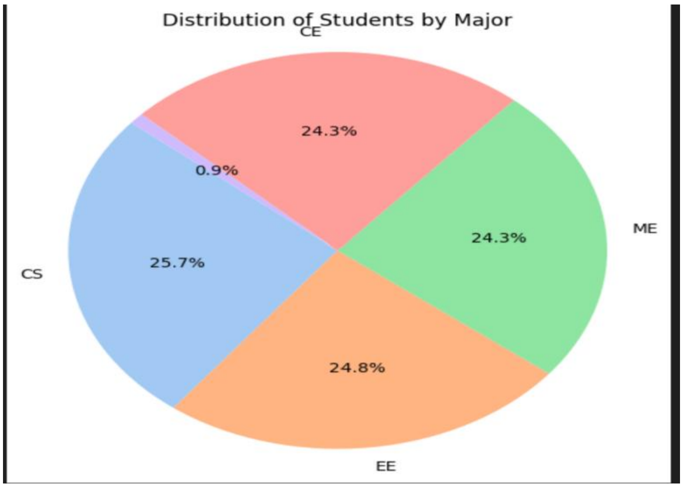












TESTING And Bonus Features

GUI Testing for Student Information Management System (SIMS)

GUI testing is a critical phase in the development of our Student Information Management System (SIMS), ensuring that the graphical user interface functions flawlessly and meets user expectations. Our testing strategy encompasses several key aspects to guarantee the reliability, functionality, and usability of the interface.

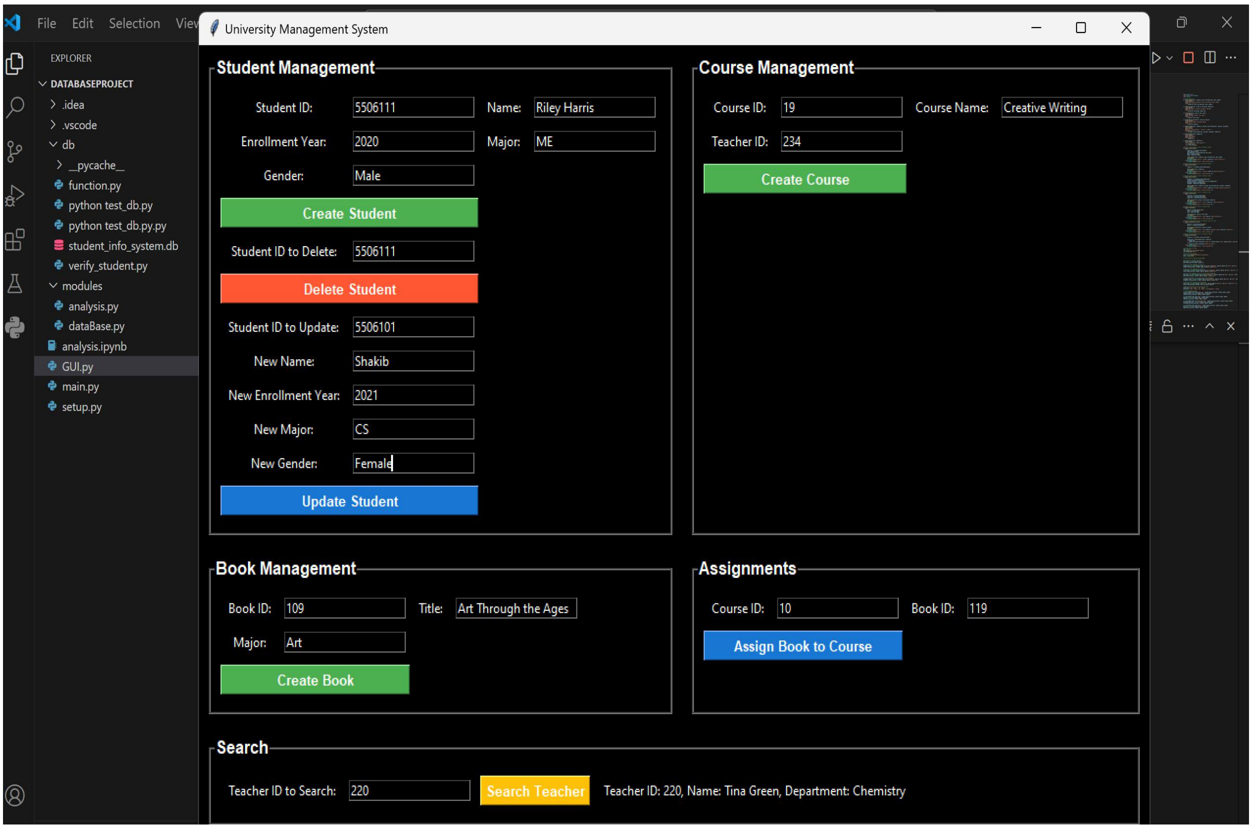
Firstly, we conducted Component Verification to ensure that each GUI element—buttons, input fields, labels, and their alignment—appears correctly and maintains consistency across different screen resolutions and devices. This step is crucial in providing a visually cohesive experience for users.

Functionality Testing was rigorous, encompassing every interactive element to verify that they perform their intended operations accurately. This includes creating, updating, and deleting student records, managing courses, assigning books, and handling various user inputs effectively.

Error Handling mechanisms were thoroughly tested to ensure the system responds appropriately to invalid inputs or exceptional conditions. Tkinter's messagebox module was utilized to display clear and informative error messages, guiding users through corrective actions without disrupting their workflow.

Usability and Navigation within the GUI were evaluated to ensure intuitive user interaction and logical flow between different screens and functionalities. This enhances user experience by making operations straightforward and accessible.

Snapshot of GUI:



Student ID: 5506111 Name: Riley Harris
Enrollment Year: 2020 Major: ME
Gender: Male

Student ID to Delete: 5506111

Student ID to Update: 5506101
New Name: Shakib
New Enrollment Year: 2021
New Major: CS
New Gender: Female

Course ID: 19 Course Name: Creative Writing
Teacher ID: 234

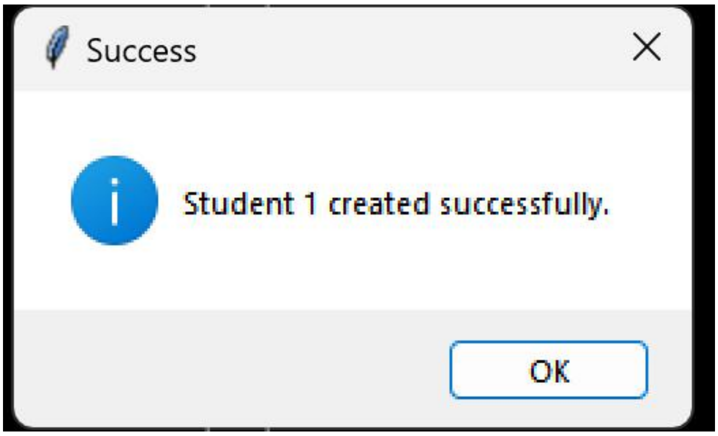
Book ID: 109 Title: Art Through the Ages
Major: Art

Course ID: 10 Book ID: 119

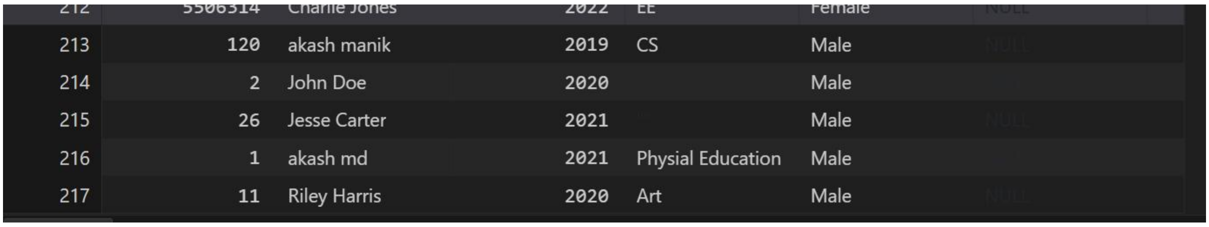
Teacher ID to Search: 220

Teacher ID: 220, Name: Tina Green, Department: Chemistry

Create Student:

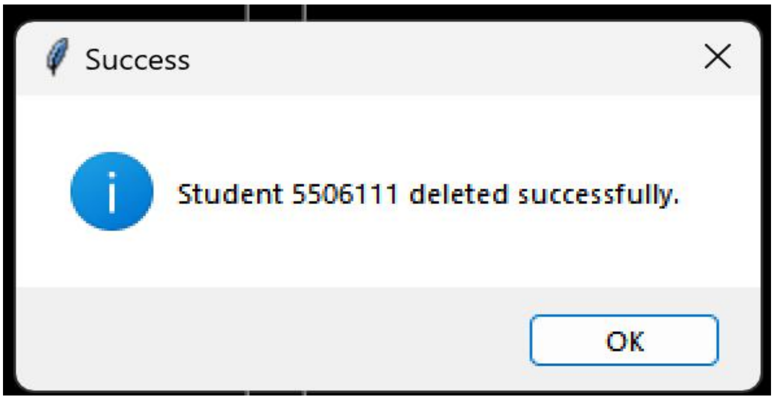


Student 1 created successfully.

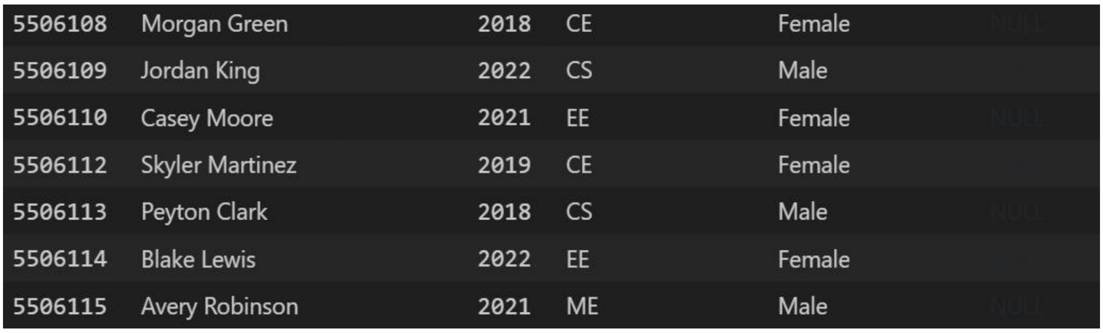


212	5506314	Charlie Jones	2022	EE	Female	NULL
213	120	akash manik	2019	CS	Male	NULL
214	2	John Doe	2020		Male	
215	26	Jesse Carter	2021		Male	NULL
216	1	akash md	2021	Physial Education	Male	
217	11	Riley Harris	2020	Art	Male	NULL

Delete Student:

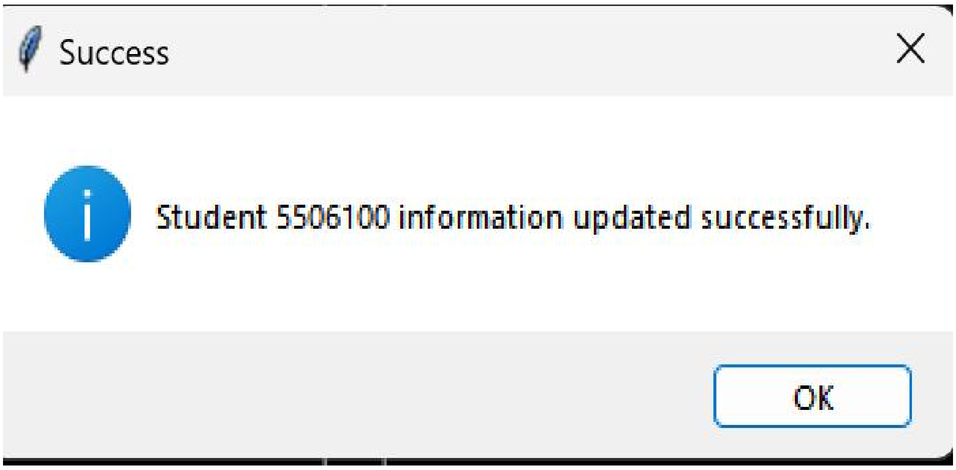


Student 5506111 deleted successfully.

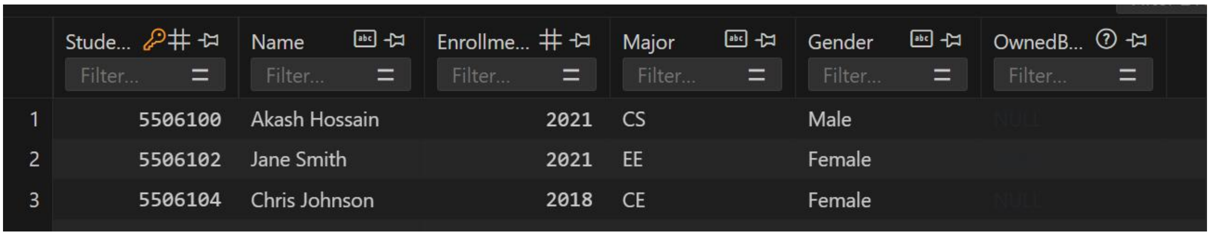


5506108	Morgan Green	2018	CE	Female	NULL
5506109	Jordan King	2022	CS	Male	
5506110	Casey Moore	2021	EE	Female	NULL
5506112	Skyler Martinez	2019	CE	Female	
5506113	Peyton Clark	2018	CS	Male	NULL
5506114	Blake Lewis	2022	EE	Female	
5506115	Avery Robinson	2021	ME	Male	NULL

Update student:

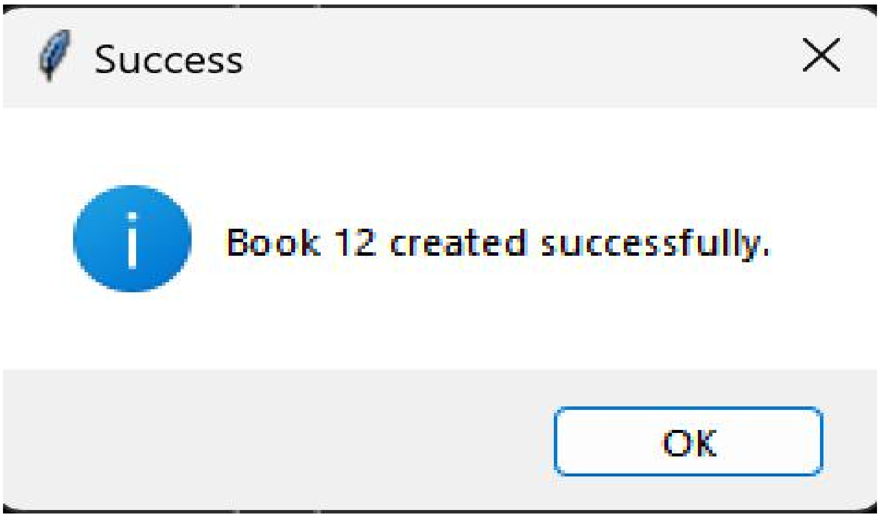


Student 5506100 information updated successfully.



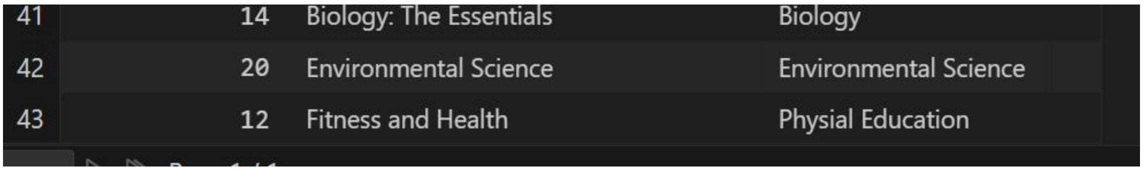
	Stude...	Name	Enrollme...	Major	Gender	OwnedB...
1	5506100	Akash Hossain	2021	CS	Male	NULL
2	5506102	Jane Smith	2021	EE	Female	
3	5506104	Chris Johnson	2018	CE	Female	NULL

Create Book:



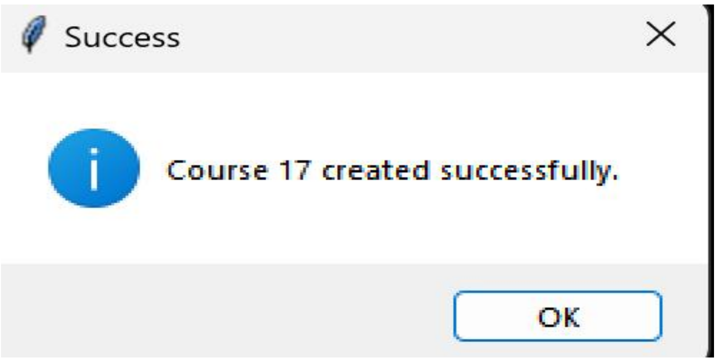
Success

Book 12 created successfully.



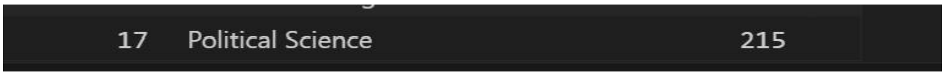
41	14	Biology: The Essentials	Biology
42	20	Environmental Science	Environmental Science
43	12	Fitness and Health	Physial Education

Create Course:



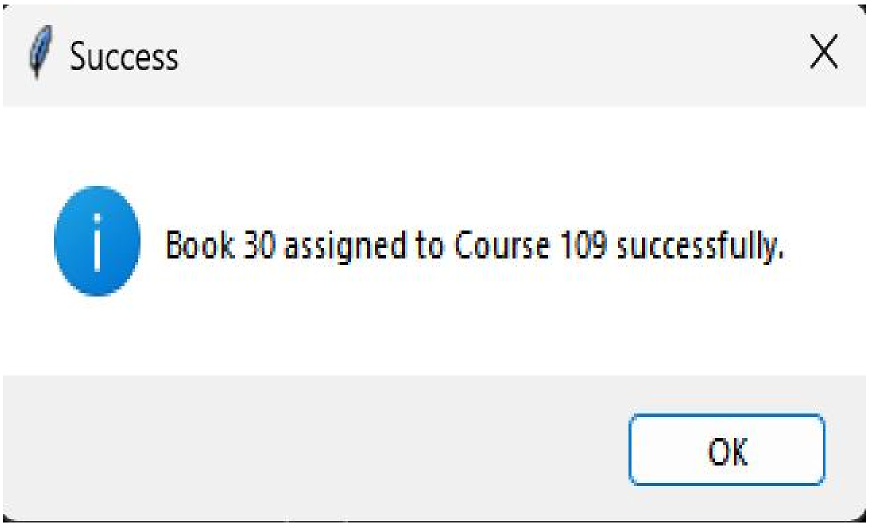
Success

Course 17 created successfully.

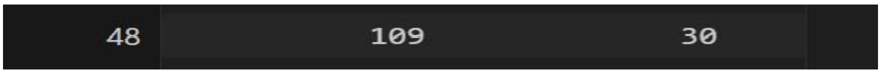


17	Political Science	215

Assign Book to Course:

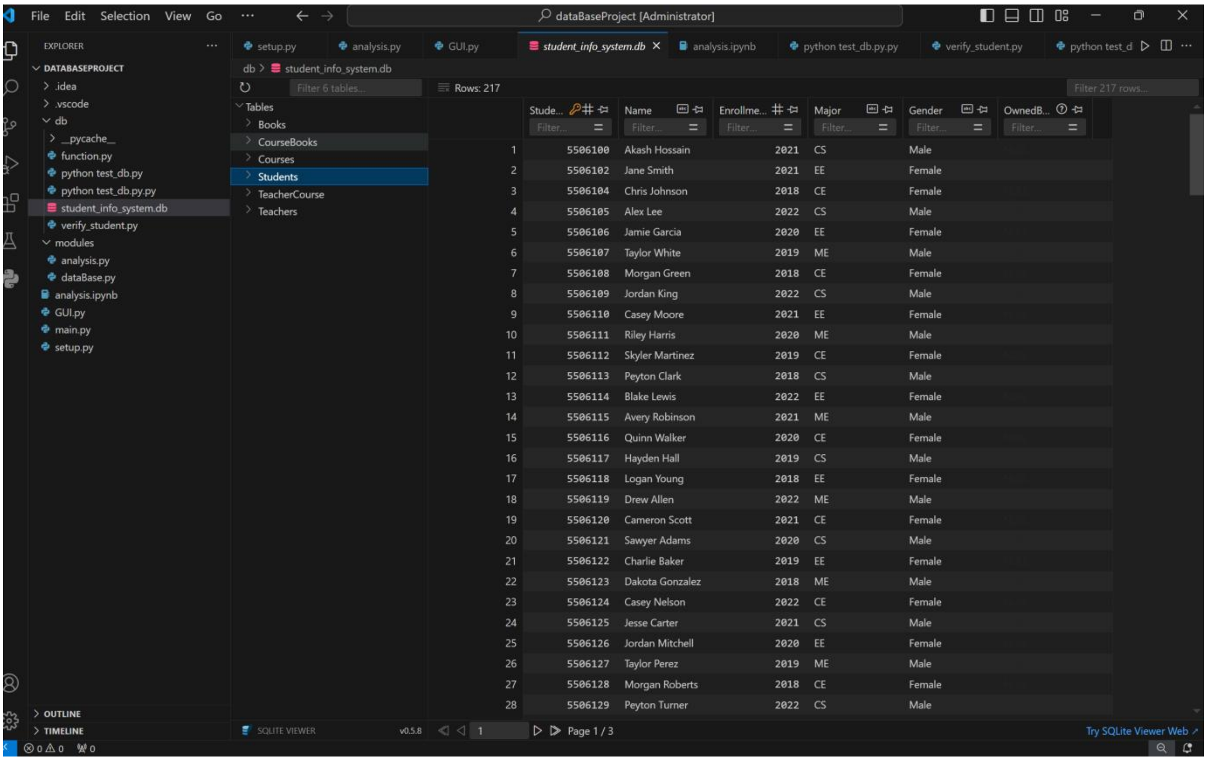


Book 30 assigned to Course 109 successfully.



48	109	30

Snapshot Of Create table :



	Stude...	Name	Enrollme...	Major	Gender	OwnedB...
1	5506100	Akash Hossain	2021	CS	Male	
2	5506102	Jane Smith	2021	EE	Female	
3	5506104	Chris Johnson	2018	CE	Female	
4	5506105	Alex Lee	2022	CS	Male	
5	5506106	Jamie Garcia	2020	EE	Female	
6	5506107	Taylor White	2019	ME	Male	
7	5506108	Morgan Green	2018	CE	Female	
8	5506109	Jordan King	2022	CS	Male	
9	5506110	Casey Moore	2021	EE	Female	
10	5506111	Riley Harris	2020	ME	Male	
11	5506112	Skyler Martinez	2019	CE	Female	
12	5506113	Peyton Clark	2018	CS	Male	
13	5506114	Blake Lewis	2022	EE	Female	
14	5506115	Avery Robinson	2021	ME	Male	
15	5506116	Quinn Walker	2020	CE	Female	
16	5506117	Hayden Hall	2019	CS	Male	
17	5506118	Logan Young	2018	EE	Female	
18	5506119	Drew Allen	2022	ME	Male	
19	5506120	Cameron Scott	2021	CE	Female	
20	5506121	Sawyer Adams	2020	CS	Male	
21	5506122	Charlie Baker	2019	EE	Female	
22	5506123	Dakota Gonzalez	2018	ME	Male	
23	5506124	Casey Nelson	2022	CE	Female	
24	5506125	Jesse Carter	2021	CS	Male	
25	5506126	Jordan Mitchell	2020	EE	Female	
26	5506127	Taylor Perez	2019	ME	Male	
27	5506128	Morgan Roberts	2018	CE	Female	
28	5506129	Peyton Turner	2022	CS	Male	

Building the Database and Test Code for Student Information Management System

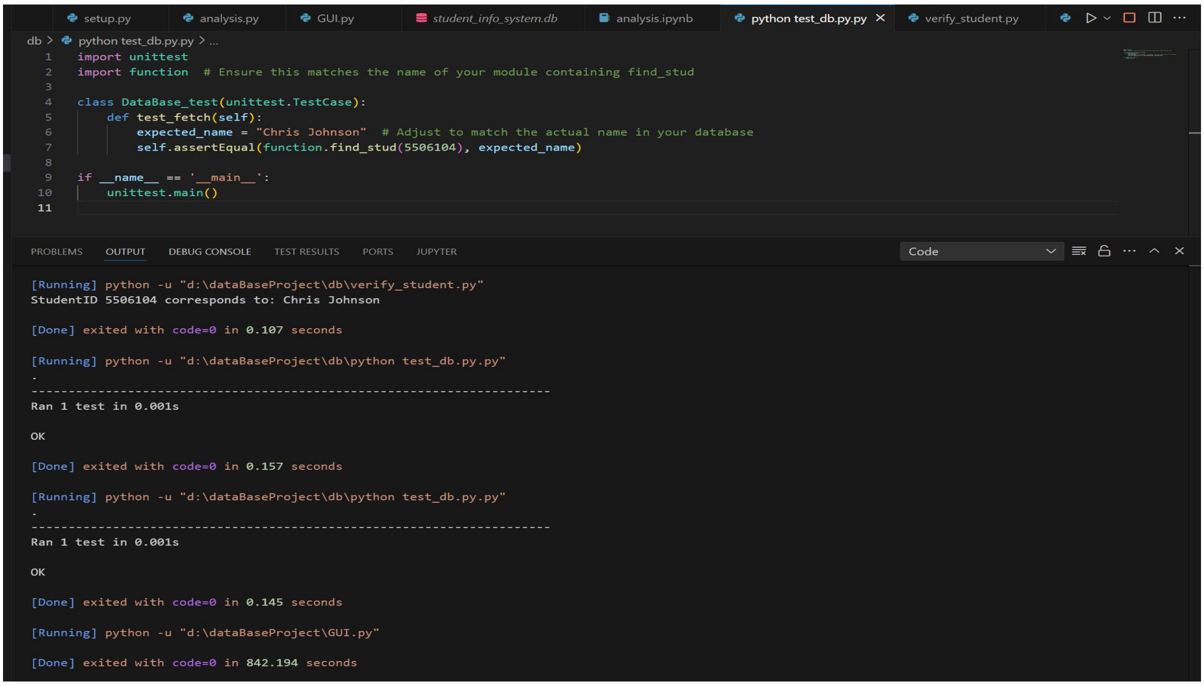
The provided code is designed to verify and validate the retrieval of a student's name from a SQLite database based on their StudentID. It comprises two main components: a unit test for automatic validation and a function for manual verification.

The unit test uses Python's unittest framework to ensure the correctness of the find_stud function. This function queries the database and returns the student's name for a given StudentID. The test checks if the function correctly retrieves "Chris Johnson" for StudentID 5506104. The test case is executed by calling unittest.main(), which runs all defined test methods and reports the results.

The verify_student function manually connects to the SQLite database located at D:\dataBaseProject\db\student_info_system.db. It executes a SQL query to fetch the name of the student with StudentID 5506104 and prints the result. If the student is found, their name is displayed; otherwise, a message indicating no match is printed. Error handling is included to catch and report any database connection issues, ensuring robustness.

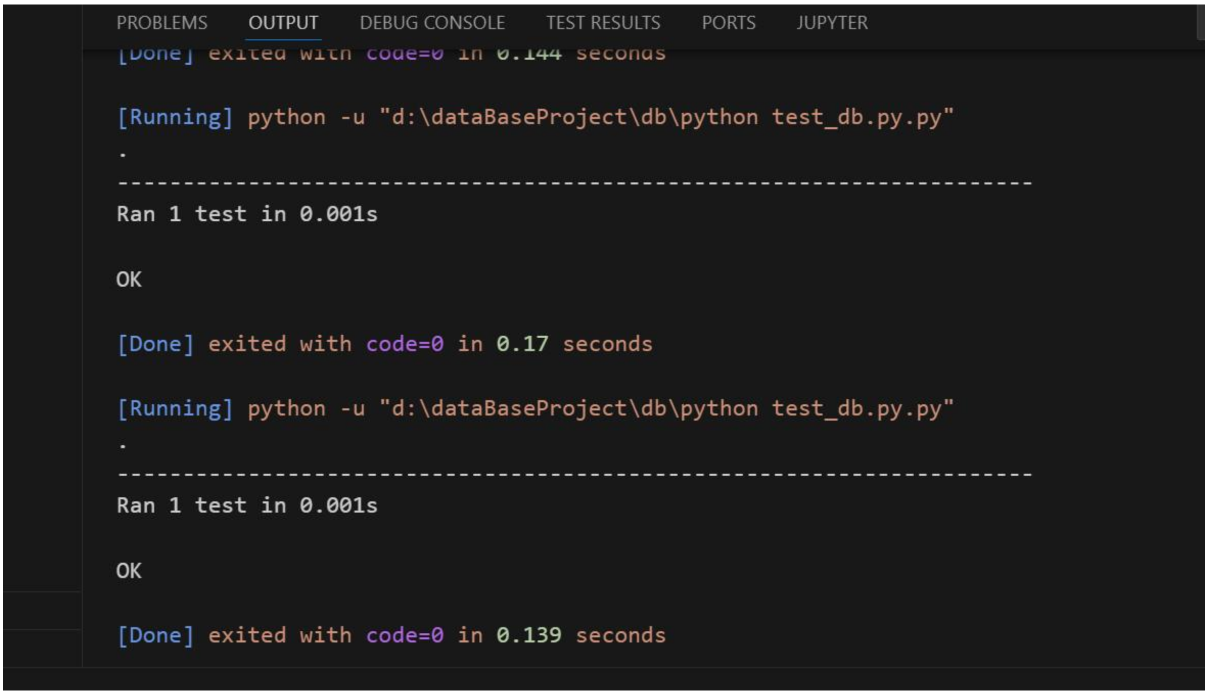
Together, these components provide a comprehensive approach to validating database queries. The unit test offers automated validation to ensure the function works as expected, while the verification function allows for manual inspection of the database contents. This dual approach ensures both accuracy and reliability in database operations, crucial for maintaining data integrity in the Student Information Management System (SIMS).

Snapshot of python test_db.py

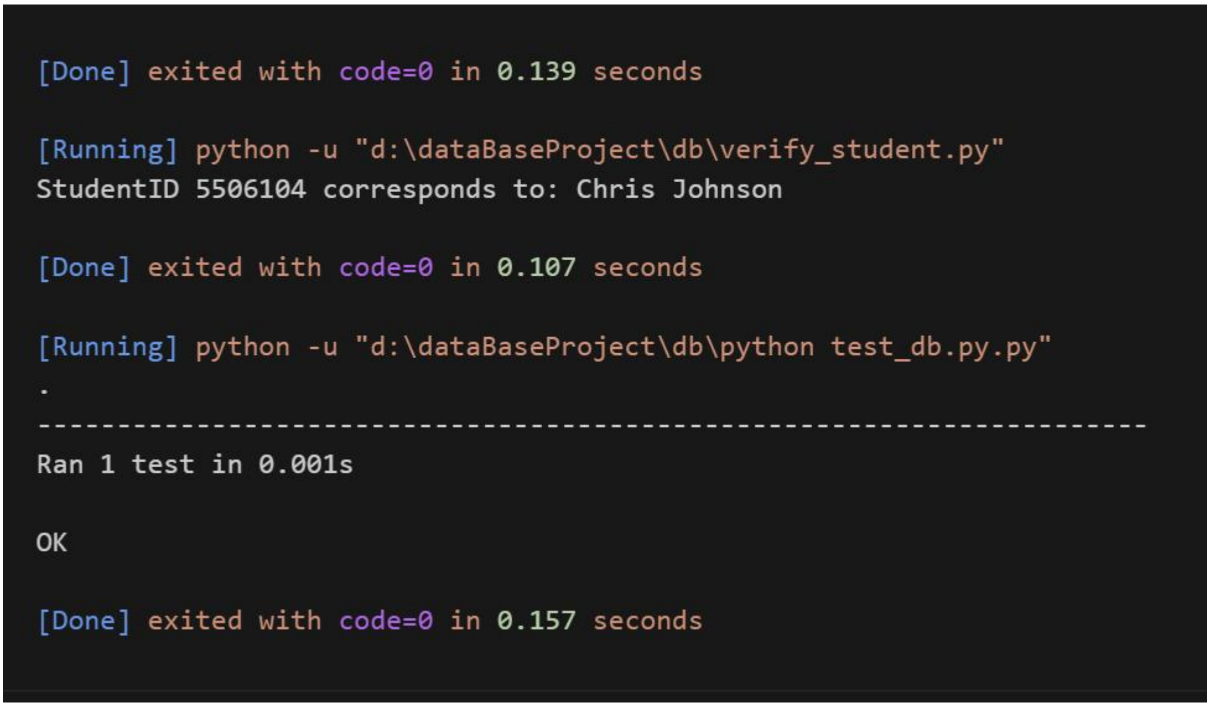


```python
import unittest
import function  # Ensure this matches the name of your module containing find_stud

class DataBase_test(unittest.TestCase):
    def test_fetch(self):
        expected_name = "Chris Johnson"  # Adjust to match the actual name in your database
        self.assertEqual(function.find_stud(5506104), expected_name)

if __name__ == '__main__':
    unittest.main()

```

```
[Running] python -u "d:\dataBaseProject\db\verify_student.py"
StudentID 5506104 corresponds to: Chris Johnson

[Done] exited with code=0 in 0.107 seconds

[Running] python -u "d:\dataBaseProject\db\python test_db.py.py"
.
----------------------------------------------------------------------
Ran 1 test in 0.001s

OK

[Done] exited with code=0 in 0.157 seconds

[Running] python -u "d:\dataBaseProject\db\python test_db.py.py"
.
----------------------------------------------------------------------
Ran 1 test in 0.001s

OK

[Done] exited with code=0 in 0.145 seconds

[Running] python -u "d:\dataBaseProject\GUI.py"

[Done] exited with code=0 in 842.194 seconds
```



```
[Done] exited with code=0 in 0.144 seconds

[Running] python -u "d:\dataBaseProject\db\python test_db.py.py"
.
----------------------------------------------------------------------
Ran 1 test in 0.001s

OK

[Done] exited with code=0 in 0.17 seconds

[Running] python -u "d:\dataBaseProject\db\python test_db.py.py"
.
----------------------------------------------------------------------
Ran 1 test in 0.001s

OK

[Done] exited with code=0 in 0.139 seconds
```



```
[Done] exited with code=0 in 0.139 seconds

[Running] python -u "d:\dataBaseProject\db\verify_student.py"
StudentID 5506104 corresponds to: Chris Johnson

[Done] exited with code=0 in 0.107 seconds

[Running] python -u "d:\dataBaseProject\db\python test_db.py.py"
.
----------------------------------------------------------------------
Ran 1 test in 0.001s

OK

[Done] exited with code=0 in 0.157 seconds
```

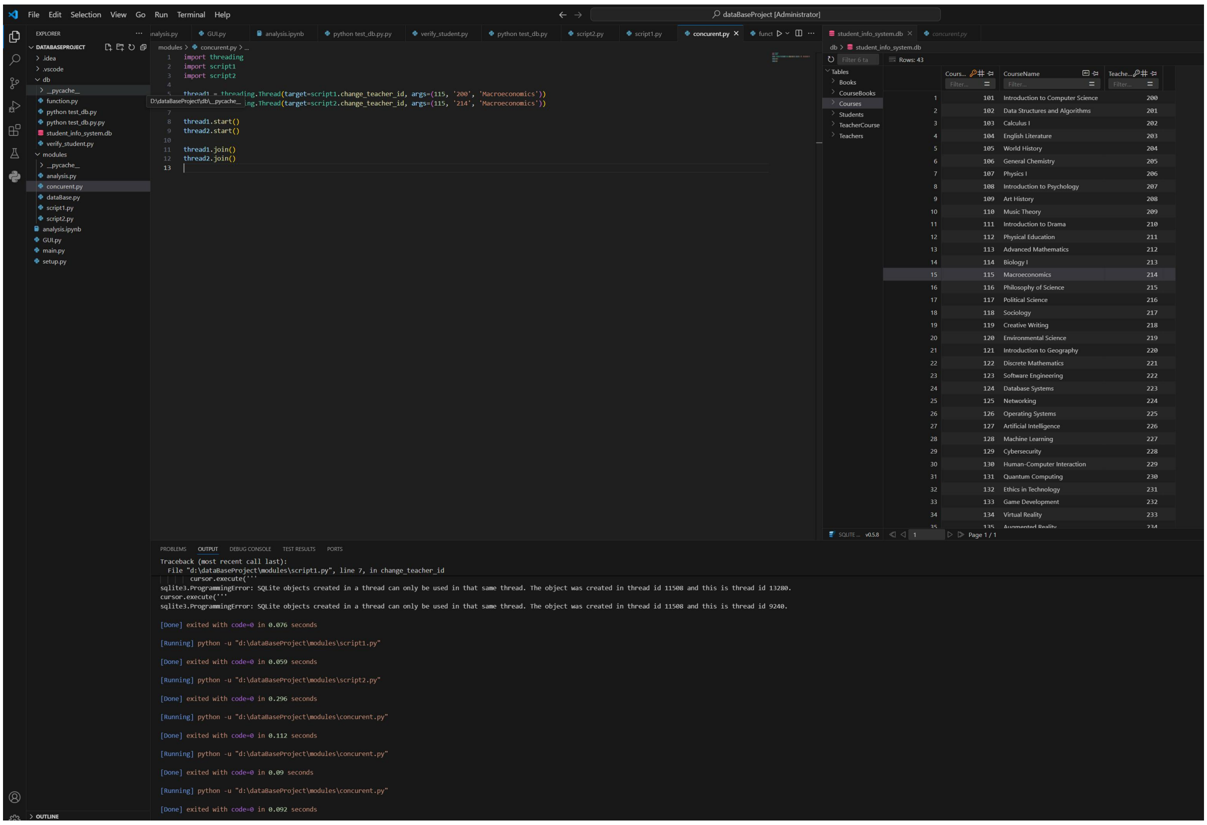
Concurrent Query Tests

The concurrent query tests for the Student Information Management System (SIMS) aimed to evaluate how SQLite handles simultaneous updates. The Courses table, specifically the TeacherID for "Macroeconomics" (CourseID 115), was updated concurrently by two scripts. Each script established its own database connection to comply with SQLite’s threading model. The concurrent execution was initiated by concurrent.py, which ran script1.py and script2.py in separate threads.

The tests resulted in a race condition, where the final TeacherID was either 200 or 116, depending on which thread's update was committed last. This highlighted SQLite's limitations in handling concurrent updates, as the lightweight database is not optimized for high-concurrency scenarios without additional mechanisms.

To address these issues, it is recommended to implement transaction management and row-level locking to ensure atomicity and isolation of updates. Alternatively, switching to more robust database systems like PostgreSQL or MySQL can provide better support for concurrent transactions. Refactoring the update functions to handle conflicts and retries can further enhance data integrity and consistency in multi-threaded environments.

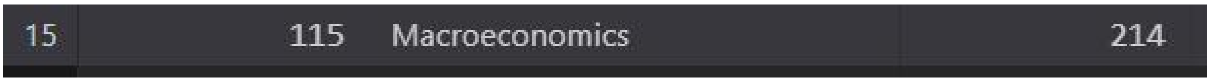
Snapshot of Before concurrent



```python
import threading
import script1
import script2

thread1 = threading.Thread(target=script1.change_teacher_id, args=(115, '200', 'Macroeconomics'))
D:\dataBaseProject\db\__pycache__ ing.Thread(target=script2.change_teacher_id, args=(115, '214', 'Macroeconomics'))

thread1.start()
thread2.start()

thread1.join()
thread2.join()
```

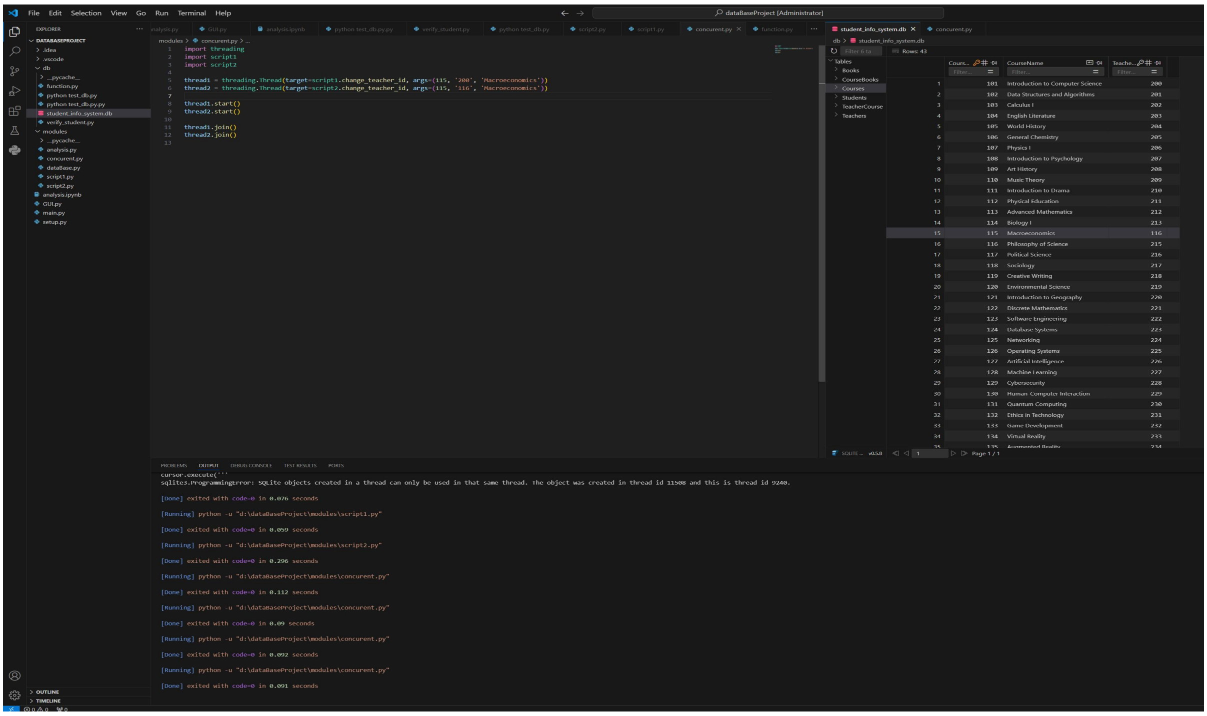
```
Traceback (most recent call last):
  File "d:\dataBaseProject\modules\script1.py", line 7, in change_teacher_id
        cursor.execute('''
sqlite3.ProgrammingError: SQLite objects created in a thread can only be used in that same thread. The object was created in thread id 11508 and this is thread id 13280.
cursor.execute('''
sqlite3.ProgrammingError: SQLite objects created in a thread can only be used in that same thread. The object was created in thread id 11508 and this is thread id 9240.

[Done] exited with code=0 in 0.076 seconds

[Running] python -u "d:\dataBaseProject\modules\script1.py"

[Done] exited with code=0 in 0.059 seconds

[Running] python -u "d:\dataBaseProject\modules\script2.py"

[Done] exited with code=0 in 0.296 seconds

[Running] python -u "d:\dataBaseProject\modules\concurent.py"

[Done] exited with code=0 in 0.112 seconds

[Running] python -u "d:\dataBaseProject\modules\concurent.py"

[Done] exited with code=0 in 0.09 seconds

[Running] python -u "d:\dataBaseProject\modules\concurent.py"

[Done] exited with code=0 in 0.092 seconds
```

	Cours...	CourseName	Teache...
1	101	Introduction to Computer Science	200
2	102	Data Structures and Algorithms	201
3	103	Calculus I	202
4	104	English Literature	203
5	105	World History	204
6	106	General Chemistry	205
7	107	Physics I	206
8	108	Introduction to Psychology	207
9	109	Art History	208
10	110	Music Theory	209
11	111	Introduction to Drama	210
12	112	Physical Education	211
13	113	Advanced Mathematics	212
14	114	Biology I	213
15	115	Macroeconomics	214
16	116	Philosophy of Science	215
17	117	Political Science	216
18	118	Sociology	217
19	119	Creative Writing	218
20	120	Environmental Science	219
21	121	Introduction to Geography	220
22	122	Discrete Mathematics	221
23	123	Software Engineering	222
24	124	Database Systems	223
25	125	Networking	224
26	126	Operating Systems	225
27	127	Artificial Intelligence	226
28	128	Machine Learning	227
29	129	Cybersecurity	228
30	130	Human-Computer Interaction	229
31	131	Quantum Computing	230
32	132	Ethics in Technology	231
33	133	Game Development	232
34	134	Virtual Reality	233
35	135	Augmented Reality	234



15	115	Macroeconomics	214

Snapshot of After concurrent



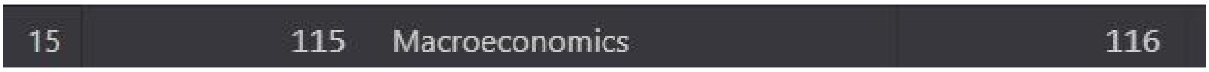
```python
import threading
import script1
import script2

thread1 = threading.Thread(target=script1.change_teacher_id, args=(115, '200', 'Macroeconomics'))
thread2 = threading.Thread(target=script2.change_teacher_id, args=(115, '116', 'Macroeconomics'))

thread1.start()
thread2.start()

thread1.join()
thread2.join()
```

```
cursor.execute('''
sqlite3.ProgrammingError: SQLite objects created in a thread can only be used in that same thread. The object was created in thread id 11508 and this is thread id 9240.

[Done] exited with code=0 in 0.076 seconds

[Running] python -u "d:\dataBaseProject\modules\script1.py"

[Done] exited with code=0 in 0.059 seconds

[Running] python -u "d:\dataBaseProject\modules\script2.py"

[Done] exited with code=0 in 0.296 seconds

[Running] python -u "d:\dataBaseProject\modules\concurent.py"

[Done] exited with code=0 in 0.112 seconds

[Running] python -u "d:\dataBaseProject\modules\concurent.py"

[Done] exited with code=0 in 0.09 seconds

[Running] python -u "d:\dataBaseProject\modules\concurent.py"

[Done] exited with code=0 in 0.092 seconds

[Running] python -u "d:\dataBaseProject\modules\concurent.py"

[Done] exited with code=0 in 0.091 seconds
```

Rows: 43

	Cours...	CourseName	Teache...
1	101	Introduction to Computer Science	200
2	102	Data Structures and Algorithms	201
3	103	Calculus I	202
4	104	English Literature	203
5	105	World History	204
6	106	General Chemistry	205
7	107	Physics I	206
8	108	Introduction to Psychology	207
9	109	Art History	208
10	110	Music Theory	209
11	111	Introduction to Drama	210
12	112	Physical Education	211
13	113	Advanced Mathematics	212
14	114	Biology I	213
15	115	Macroeconomics	116
16	116	Philosophy of Science	215
17	117	Political Science	216
18	118	Sociology	217
19	119	Creative Writing	218
20	120	Environmental Science	219
21	121	Introduction to Geography	220
22	122	Discrete Mathematics	221
23	123	Software Engineering	222
24	124	Database Systems	223
25	125	Networking	224
26	126	Operating Systems	225
27	127	Artificial Intelligence	226
28	128	Machine Learning	227
29	129	Cybersecurity	228
30	130	Human-Computer Interaction	229
31	131	Quantum Computing	230
32	132	Ethics in Technology	231
33	133	Game Development	232
34	134	Virtual Reality	233
35	135	Augmented Reality	234

Page 1 / 1



15	115	Macroeconomics	116

CONCLUSION

The Student Information Management System (SIMS) project aims to revolutionize the way educational institutions manage student data by transitioning from manual to automated systems. This project addresses the inefficiencies and inaccuracies inherent in traditional paper-based systems, providing a centralized platform for managing student records, teacher assignments, course details, and associated resources.

The SIMS database is designed using SQLite, incorporating tables for Students, Teachers, Courses, and Books, each with unique identifiers and relevant attributes. This structure ensures data integrity and supports complex relationships, such as linking teachers to courses and books to courses. CRUD (Create, Read, Update, Delete) operations are implemented to manage these records efficiently, ensuring that data is consistently accurate and up-to-date.

Data analysis and visualization are key components of SIMS, offering insights into student demographics, enrollment trends, teacher workloads, and resource allocation. Functions like studentNumber() and studentCountByYear() provide snapshots of the student population and enrollment trends, while courseCountByTeacher() and teachers With Many Courses(minCourses) help balance teaching responsibilities and optimize course assignments.

The system's GUI, developed using Tkinter, ensures a user-friendly interface for administrators, teachers, and students. Rigorous GUI testing ensures that all elements function correctly and provide a seamless user experience. Features like creating and updating student records, assigning books to courses, and managing teacher assignments are all accessible through this interface.

Security and data integrity are paramount, with measures such as access controls and data validation in place to protect sensitive information. The system's modular design allows for future enhancements, such as advanced reporting features, mobile application integration, and more robust security protocols.

In conclusion, the SIMS project successfully automates and centralizes student information management, enhancing operational efficiency, data accuracy, and user experience in educational institutions. This system not only streamlines administrative tasks but also provides valuable insights for decision-making, ultimately contributing to a more effective and responsive educational environment.