ADVANCE DATABASE SYSTEM DESIGN

Class Project

Name: Tirumalasetty Naga Sudha Pavani

Student Id: 811294971

Student Course Management System:

The Student Course Management System is a web-based application designed to organize the management of students and course details within an educational institution. It allows administrators to perform essential CRUD operations such as creating, adding, updating, and deleting on student and course records. The system provides functionalities like viewing lists of students and courses and the corresponding Professors, adding new students, updating existing student details, managing course information such as course name, and Professor name facilitating deletion of student or course records. Additionally, the system includes a search feature enabling users to find specific students or courses efficiently. The application is built using the Bottle framework in Python and utilizes SQLite as its database backend. The HTML templates define the user interface for adding, updating, and displaying lists of students and courses. The Python code handles routing, form submissions, database interactions, and CRUD operations (Create, Read, Update, Delete) for students and courses. Overall, it provides a simple interface to manage student and course information stored in a SQLite database.

The Student Course Management System offers several benefits:

1) It helps to manage student and course details efficiently by storing it in an organized manner within a SQLite database. This reduces manual efforts in managing records and ensures accuracy.It organizes student and course data systematically, making it easier to retrieve specific information quickly.
2) The application allows CRUD operations (Create, Read, Update, Delete) on both students and course details. Users can perform all essential operations to manage and maintain the database.
3) Users can search for specific students or courses using a search query, making it convenient to locate information within a large dataset.
4) The project demonstrates a structured approach to building a web application, separating concerns between the front-end (HTML templates) and back-end (Python/Bottle framework and SQLite database).

Tables/Collection:

A Student Course Management System requires at least two tables or collections for effective data organization and management:

1) **Student Table:** This table/collection stores information about students, such as their names, unique identifiers. This table allows for individual student management, including addition, update, deletion, and retrieval of student information.

2) **Course Table:** This table/collection holds details about the courses available, such as course names, Professor name. This table allows for managing courses independently, including adding, updating, deleting, and retrieving course information.

The need for two separate tables or collections is due to the relationship between students and courses records:

- In an educational context, many students can enroll in multiple courses, and a single course can have multiple students. This forms a many-to-many relationship between students and courses.
  To represent this relationship, we need a separate table, this typically includes student IDs and course IDs, linking students to the courses they are enrolled in.
- Using separate tables allows for the normalization of the database. Each table contains specific types of information, and this separation reduces data redundancy and improves data integrity.
  For instance, instead of storing the professor's name in the student table, the courses table specifically manages course-related details, and the students table manages student-related details.
- Using at least two tables facilitates the organization, normalization, and efficient management of data, in situations like complex relationships, such as the many-to-many relationship between students and course records.

Database and Access Methods:

1) CRUD Operations and SQL Queries:
   - Reasoning: CRUD operations (Create, Read, Update, Delete) are fundamental for managing data in the application. SQLite's support allows execution of these operations.
   - Consideration: SQL queries are used extensively to perform database actions like adding students/courses, updating their details, fetching specific records, or deleting entries.

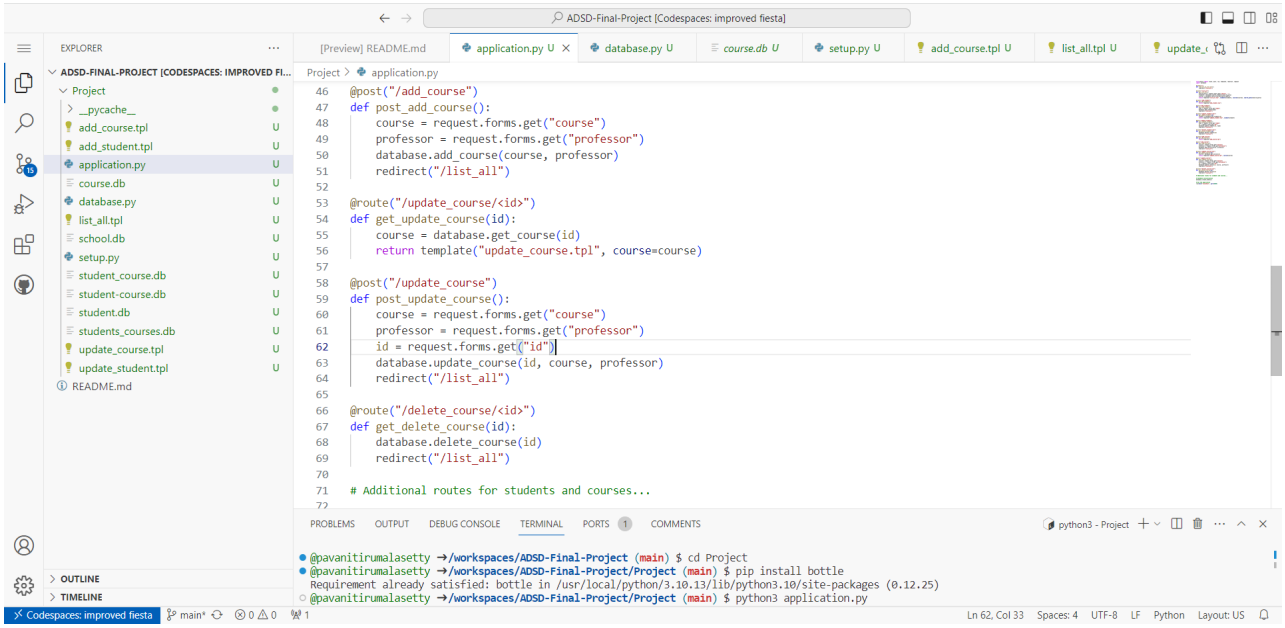
2) SQLite Database:
   - Reasoning: SQLite is chosen for its ease of setup. It's a serverless database that operates using a single file, making it convenient for smaller-scale applications or prototypes.
   - Consideration: It helps in managing students and course details, SQLite provides adequate functionality without the need for a dedicated database server.

3) Normalized Data Structure:
   - Reasoning: To structure the database with separate tables for students and courses helps in normalization principles, minimizing data redundancy and maintaining data integrity.
   - Consideration: This structured approach helps for efficient data management, especially when dealing with relationships between students and courses (many-to-many).

Code for Student Course Management System:

The below are the codes used for the creating student and course tables.

Web Application code:



add_course. tpl:

```
<html>
<body>
<h2>Add a New Course</h2>
<hr/>

<form action="/add_course" method="post">
  <p>Course Name: <input name="course" required/></p>
  <p>Professor: <input name="professor" required/></p>
  <p><button type="submit">Submit</button></p>
</form>

<hr/>
<a href="/list_all">Back to Courses List</a>
<hr/>
</body>
</html>
```

add_student.tpl:

```
<html>
<body>
<h2>Add a New Student</h2>
<hr/>
```

```
<form action="/add_student" method="post">
  <p>Student Name: <input name="name" required/></p>
  <p><button type="submit">Submit</button></p>
</form>

<hr/>
<a href="/list_all">Back to Students List</a>
<hr/>
</body>
</html>
```

update_course.tpl:

```
<html>
<body>
<hr/>
<form action="/update_course" method="post">
  <input type="hidden" name="id" value="{{ course['id'] }}"/>
  <p>Course Name: <input name="course" value="{{ course['course'] }}"/></p>
  <p>Professor: <input name="professor" value="{{ course['professor'] }}"/></p>
  <p><button type="submit">Update</button></p>
</form>
<hr/>
</body>
</html>
```

update_student.tpl:

```
<html>
<body>
<hr/>
<form action="/update_student" method="post">
  <input type="hidden" name="id" value="{{ student['id'] }}"/>
  <p>Student Name: <input name="name" value="{{ student['name'] }}"/></p>
  <p><button type="submit">Update</button></p>
</form>
<hr/>
</body>
</html>
```

list_all.tpl:

```
<html>
<body>
<form action="/list_all" method="get">
    <label for="search">Search:</label>
```

```
    <input type="text" name="search" value="{{search_query}}">
    <button type="submit">Submit</button>
</form>
<h2>Students Details</h2>
 <hr/>
  <table>
    <tr>
      <th>Student ID</th>
      <th>Student Name</th>
      <th>Actions</th>
    </tr>
    % for student in students:
      <tr>
        <td>{{ student['id'] }}</td>
        <td>{{ student['name'] }}</td>
        <td>
          <a href="/update_student/{{ student['id'] }}">update</a>
          <a href="/delete_student/{{ student['id'] }}">delete</a>
        </td>
      </tr>
    % end
  </table>
  <hr/>
  <a href="/add_student">Add a new student</a>
  <hr/>

  <h2>Course Details</h2>
  <hr/>
  <table>
    <tr>
      <th>Course ID</th>
      <th>Course Name</th>
      <th>Professor</th>
      <th>Actions</th>
    </tr>
    % for course in courses:
      <tr>
        <td>{{ course['id'] }}</td>
        <td>{{ course['course'] }}</td>
        <td>{{ course['professor'] }}</td>
        <td>
          <a href="/update_course/{{ course['id'] }}">update</a>
          <a href="/delete_course/{{ course['id'] }}">delete</a>
        </td>
```

```
    </tr>
  % end
 </table>
 <hr/>
 <a href="/add_course">Add a new course</a>
 <hr/>
</body>
</html>
```

application.py:

```
from bottle import route, post, run, template, redirect, request
import database

@route("/")
def redirect_to_list_all():
    redirect("/list_all")

@route("/list_all")
def get_list_all():
    search_query = request.query.get('search', '')
    students = database.search_students(search_query)
    courses = database.search_courses(search_query)
    return template("list_all.tpl", students=students, courses=courses, search_query=search_query)

@route("/add_student")
def get_add_student():
    return template("add_student.tpl")

@post("/add_student")
def post_add_student():
    name = request.forms.get("name")
    database.add_student(name)
    redirect("/list_all")

@route("/update_student/<id>")
def get_update_student(id):
    student = database.get_student(id)
    return template("update_student.tpl", student=student)
```

```
@post("/update_student")
def post_update_student():
    name = request.forms.get("name")
    id = request.forms.get("id")
    database.update_student(id, name)
    redirect("/list_all")

@route("/delete_student/<id>")
def get_delete_student(id):
    database.delete_student(id)
    redirect("/list_all")

@route("/add_course")
def get_add_course():
    return template("add_course.tpl")

@post("/add_course")
def post_add_course():
    course = request.forms.get("course")
    professor = request.forms.get("professor")
    database.add_course(course, professor)
    redirect("/list_all")

@route("/update_course/<id>")
def get_update_course(id):
    course = database.get_course(id)
    return template("update_course.tpl", course=course)

@post("/update_course")
def post_update_course():
    course = request.forms.get("course")
    professor = request.forms.get("professor")
    id = request.forms.get("id")
    database.update_course(id, course, professor)
    redirect("/list_all")

@route("/delete_course/<id>")
def get_delete_course(id):
```

```
    database.delete_course(id)
    redirect("/list_all")

# Additional routes for students and courses...

# Database Initialization
database.create_tables()

# Run the application
run(host='localhost', port=8080)
```

database.py:

```
import sqlite3

connection = sqlite3.connect("students_courses.db")
cursor = connection.cursor()

def create_tables():
    cursor.execute("CREATE TABLE IF NOT EXISTS students (id INTEGER PRIMARY KEY, name TEXT)")
    cursor.execute("CREATE TABLE IF NOT EXISTS courses (id INTEGER PRIMARY KEY, course TEXT, professor TEXT)")
    connection.commit()

def add_student(name):
    cursor.execute("INSERT INTO students (name) VALUES (?)", (name,))
    connection.commit()

def get_student(student_id):
    rows = cursor.execute("SELECT id, name FROM students WHERE id=?", (student_id,))
    rows = list(rows)
    if rows:
        return {'id': rows[0][0], 'name': rows[0][1]}
    else:
        return None

def update_student(student_id, name):
```

```
    cursor.execute("UPDATE students SET name=? WHERE id=?", (name, student_id))
    connection.commit()

def delete_student(student_id):
    cursor.execute("DELETE FROM students WHERE id=?", (student_id,))
    connection.commit()

def search_students(query):
    query = f"%{query}%"
    rows = cursor.execute("SELECT id, name FROM students WHERE name LIKE ?", (query,))
    rows = list(rows)
    students = [{'id': row[0], 'name': row[1]} for row in rows]
    return students

def add_course(course, professor):
    cursor.execute("INSERT INTO courses (course, professor) VALUES (?, ?)", (course, professor))
    connection.commit()

def get_course(course_id):
    rows = cursor.execute("SELECT id, course, professor FROM courses WHERE id=?", (course_id,))
    rows = list(rows)
    if rows:
        return {'id': rows[0][0], 'course': rows[0][1], 'professor': rows[0][2]}
    else:
        return None

def update_course(course_id, course, professor):
        cursor.execute("UPDATE courses SET course=?, professor=? WHERE id=?", (course, professor,
course_id))
    connection.commit()

def delete_course(course_id):
    cursor.execute("DELETE FROM courses WHERE id=?", (course_id,))
    connection.commit()

def search_courses(query):
    query = f"%{query}%"
```

```
    rows = cursor.execute("SELECT id, course, professor FROM courses WHERE course LIKE ?", (query,))
    rows = list(rows)
    courses = [{'id': row[0], 'course': row[1], 'professor': row[2]} for row in rows]
    return courses
```

Setup.py:

```
import sqlite3

def create_tables():
    connection = sqlite3.connect("students_courses.db")
    cursor = connection.cursor()

    try:
        # Create students table
        cursor.execute("CREATE TABLE IF NOT EXISTS students (id INTEGER PRIMARY KEY, name TEXT)")

        # Create courses table
        cursor.execute("CREATE TABLE IF NOT EXISTS courses (id INTEGER PRIMARY KEY, course TEXT, professor TEXT)")

        # Insert sample data into students table
        students_data = [
            ('Pavani',),
            ('Sudha',),
            # Add more sample students here
        ]
        cursor.executemany("INSERT INTO students (name) VALUES (?)", students_data)

        # Insert sample data into courses table
        courses_data = [
            ('Maths', 'Tsai'),
            ('Social', 'Sarvar'),
            # Add more sample courses here
        ]
        cursor.executemany("INSERT INTO courses (course, professor) VALUES (?, ?)", courses_data)
```

```
        connection.commit()

    except Exception as e:
        print(f"Error during table creation: {e}")
    finally:
        connection.close()

if __name__ == "__main__":
    create_tables()
    print("Database setup complete.")
```

Link to the repository containing the source code:
https://github.com/pavanitirumalasetty/ADSD-Final-Project

CRUD Operation:

Search: Submit

Students Details

Student ID	Student Name	Actions
1	Pavani	update delete
2	Archana	update delete
3	Mridula	update delete
4	Lalitha	update delete

Add a new student

Course Details

Course ID	Course Name	Professor	Actions
1	Advance Database	Dr. Gregory S. DeLozier	update delete
2	Computer Science	P. Bagavandoss	update delete
3	Cryptology	Tsung-Heng Tsai	update delete
5	Robotics	Dr. Erin Bailey	update delete

Add a new course

1) **Students Details:**

- **Read:**

Search: Submit

Students Details

Student ID	Student Name	Actions
1	Pavani	update delete
2	Archana	update delete
3	Mridula	update delete
4	Lalitha	update delete

Add a new student

- **Create:** Adding Mounika name to the student details

Add a New Student

Student Name: Mounika

Submit

Back to Students List

Mounika added to the student details

Search: Submit

Students Details

Student ID	Student Name	Actions
1	Pavani	update delete
2	Archana	update delete
3	Mridula	update delete
4	Lalitha	update delete
5	Mounika	update delete

Add a new student

- **Update:** Updating Lalitha name to Manogna in student details

Student Name: Manogna

Update

Lalitha name updated to Manogna in student details

Search: Submit

Students Details

Student ID	Student Name	Actions
1	Pavani	update delete
2	Archana	update delete
3	Mridula	update delete
4	Manogna	update delete
5	Mounika	update delete

Add a new student

- **Delete:** Deleting Mridula name from student details

Students Details

Student ID	Student Name	Actions
1	Pavani	update delete
2	Archana	update delete
4	Manogna	update delete
5	Mounika	update delete

Add a new student

2) **Course details:**

- **Read:**

Course Details

Course ID	Course Name	Professor	Actions
1	Advance Database	Dr. Gregory S. DeLozier	update delete
2	Computer Science	P. Bagavandoss	update delete
3	Cryptology	Tsung-Heng Tsai	update delete
5	Robotics	Dr. Erin Bailey	update delete

Add a new course

- **Create:** Creating Course name Big Data Analytics in Course details

Add a New Course

Course Name: Big Data Analytics

Professor: Wilson Chung

Submit

Back to Courses List

Big data analytics added into course details

Course Details

Course ID	Course Name	Professor	Actions
1	Advance Database	Dr. Gregory S. DeLozier	update delete
2	Computer Science	P. Bagavandoss	update delete
3	Cryptology	Tsung-Heng Tsai	update delete
5	Robotics	Dr. Erin Bailey	update delete
6	Big Data Analytics	Wilson Chung	update delete

Add a new course

- **Update:** Updating course name Cryptology to Information System in course details

Course Name: Information System

Professor: Dr. Michael Sarver

Update

Course Details

Course ID	Course Name	Professor	Actions
1	Advance Database	Dr. Gregory S. DeLozier	update delete
2	Computer Science	P. Bagavandoss	update delete
3	Information System	Dr. Michael Sarver	update delete
5	Robotics	Dr. Erin Bailey	update delete
6	Big Data Analytics	Wilson Chung	update delete

Add a new course

Updating Professor name P. Bagavandoss to Dr. Chuck Haviland

Course Name: Computer Science

Professor: Dr. Chuck Haviland

Update

Course Details

Course ID	Course Name	Professor	Actions
1	Advance Database	Dr. Gregory S. DeLozier	update delete
2	Computer Science	Dr. Chuck Haviland	update delete
3	Information System	Dr. Michael Sarver	update delete
5	Robotics	Dr. Erin Bailey	update delete
6	Big Data Analytics	Wilson Chung	update delete

Add a new course

Delete: Deleting Robotics in course details

Course Details

Course ID	Course Name	Professor	Actions
1	Advance Database	Dr. Gregory S. DeLozier	update delete
2	Computer Science	Dr. Chuck Haviland	update delete
3	Information System	Dr. Michael Sarver	update delete
6	Big Data Analytics	Wilson Chung	update delete

Add a new course

Checking for Search Capabilities:

1) **Student Details:** Searching for Pavani name in student details

Search: Pavani Submit

Students Details

Student ID	Student Name	Actions
1	Pavani	update delete
2	Archana	update delete
4	Manogna	update delete
5	Mounika	update delete

Add a new student

After name searching-

Search: Pavani Submit

Students Details

Student ID	Student Name	Actions
1	Pavani	update delete

Add a new student

2) **Course Details:** Searching for Advanced Database in course details

Search: Advance Database Submit

Students Details

Student ID	Student Name	Actions
1	Pavani	update delete
2	Archana	update delete
4	Manogna	update delete
5	Mounika	update delete

Add a new student

Course Details

Course ID	Course Name	Professor	Actions
1	Advance Database	Dr. Gregory S. DeLozier	update delete
2	Computer Science	Dr. Chuck Haviland	update delete
3	Information System	Dr. Michael Sarver	update delete
6	Big Data Analytics	Wilson Chung	update delete

Add a new course

After course search-

Search: Advance Database Submit

Students Details

Student ID	Student Name	Actions

Add a new student

Course Details

Course ID	Course Name	Professor	Actions
1	Advance Database	Dr. Gregory S. DeLozier	update delete

Add a new course