Assignment 9

|  |  |  |
| --- | --- | --- |
| **8** | | |
| **Aim:** Create a student management system that:  1.⁠ ⁠Allows adding new students  2.⁠ ⁠Updates existing student records  3.⁠ ⁠Deletes student records  4.⁠ ⁠Displays all students  5.⁠ ⁠Searches for students by various criteria | | |
| **Code:**  import sqlite3  *# === Database Setup ===*  def create\_connection():  *"""Connect to the SQLite database."""*  try:  conn = sqlite3.connect('students.db')  return conn  except sqlite3.Error as e:  print(f"Database error: {e}")  return None  def create\_table():  *"""Create the 'students' table if it doesn't exist."""*  conn = create\_connection()  if conn:  try:  cursor = conn.cursor()  cursor.execute('''CREATE TABLE IF NOT EXISTS students (  id INTEGER PRIMARY KEY AUTOINCREMENT,  name TEXT NOT NULL,  age INTEGER,  email TEXT  )''')  conn.commit()  print("Table 'students' created successfully!")  except sqlite3.Error as e:  print(f"Table creation error: {e}")  finally:  conn.close()  *# === Core Functions ===*  def add\_student():  *"""Add a new student to the database."""*  conn = create\_connection()  name = input("Enter student name: ")  age = int(input("Enter student age: "))  email = input("Enter student email: ")  try:  cursor = conn.cursor()  cursor.execute('''INSERT INTO students (name, age, email)  VALUES (?, ?, ?)''', (name, age, email))  conn.commit()  print("Student added successfully!")  except ValueError:  print("Invalid input! Age must be a number.")  finally:  conn.close()  def update\_student():  *"""Update a student's record by ID."""*  conn = create\_connection()  student\_id = int(input("Enter student ID to update: "))  new\_name = input("Enter new name (leave blank to skip): ")  new\_age = input("Enter new age (leave blank to skip): ")  new\_email = input("Enter new email (leave blank to skip): ")  try:  cursor = conn.cursor()  updates = []  params = []  if new\_name:  updates.append("name = ?")  params.append(new\_name)  if new\_age:  updates.append("age = ?")  params.append(int(new\_age))  if new\_email:  updates.append("email = ?")  params.append(new\_email)  params.append(student\_id)  if updates:  query = f"UPDATE students SET {', '.join(updates)} WHERE id = ?"  cursor.execute(query, params)  conn.commit()  print("Student updated successfully!")  else:  print("No fields updated.")  except ValueError:  print("Invalid input!")  finally:  conn.close()  def delete\_student():  *"""Delete a student by ID."""*  conn = create\_connection()  student\_id = int(input("Enter student ID to delete: "))  try:  cursor = conn.cursor()  cursor.execute("DELETE FROM students WHERE id = ?", (student\_id,))  conn.commit()  print("Student deleted successfully!")  *# SCREENSHOT 5: Deleting a student in terminal*  except sqlite3.Error as e:  print(f"Deletion error: {e}")  finally:  conn.close()  def display\_students():  *"""Display all students."""*  conn = create\_connection()  try:  cursor = conn.cursor()  cursor.execute("SELECT \* FROM students")  students = cursor.fetchall()  print("\n=== Student List ===")  for student in students:  print(f"ID: {student[0]}, Name: {student[1]}, Age: {student[2]}, Email: {student[3]}")  except sqlite3.Error as e:  print(f"Fetch error: {e}")  finally:  conn.close()  def search\_student():  *"""Search students by name, age, or email."""*  conn = create\_connection()  search\_term = input("Search by name, age, or email: ")  try:  cursor = conn.cursor()  cursor.execute('''SELECT \* FROM students  WHERE name LIKE ? OR age = ? OR email LIKE ?''',  (f'%{search\_term}%', search\_term, f'%{search\_term}%'))  results = cursor.fetchall()  if results:  print("\n=== Search Results ===")  for student in results:  print(f"ID: {student[0]}, Name: {student[1]}, Age: {student[2]}, Email: {student[3]}")  *# SCREENSHOT 7: Search results in terminal*  else:  print("No matching records found.")  except sqlite3.Error as e:  print(f"Search error: {e}")  finally:  conn.close()  *# === Menu System ===*  def display\_menu():  print("\n===== Student Management System =====")  print("1. Add Student")  print("2. Update Student")  print("3. Delete Student")  print("4. Display All Students")  print("5. Search Student")  print("6. Exit")  def main():  create\_table()  while True:  display\_menu()  choice = input("Enter your choice (1-6): ")  if choice == '1':  add\_student()  elif choice == '2':  update\_student()  elif choice == '3':  delete\_student()  elif choice == '4':  display\_students()  elif choice == '5':  search\_student()  elif choice == '6':  print("Exiting...")  break  else:  print("Invalid choice. Try again.")  if \_\_name\_\_ == "\_\_main\_\_":  main()  **Output Screenshot:** | | |
| **Conclusion/Summary:**  This Student Management System project allowed me to apply SQLite and Python skills to create a functional CRUD application. I designed a database to store student records, implemented operations for adding/updating/deleting entries, and added search functionality. Handling errors like invalid inputs and database exceptions improved the robustness of the system. This project reinforced my understanding of database integration and user-driven interfaces. | | |
| **Student Signature & Date** | **Marks:** | **Evaluator Signature & Date** |