Attendance Management System

A PROJECT REPORT

Submitted by

Gopal Krishan Arora[Reg No:RA2311027010032]

Tanish Singh [Reg No: RA2311027010064]

Under the Guidance of

Dr.P.Rajasekar

in partial fulfillment of the requirements for the degree of
BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING
with specialization in Big Data Analytics

21CSC205P – DATABASE MANAGEMENT SYSTEMS

DEPARTMENT OF DATA SCIENCE AND BUSINESS SYSTEM
COLLEGE OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKALATHUR - 603203

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that 21CSC205P - Project report titled "**Attendance Management System**" is the bonafide work of "**GOPAL KRISHAN ARORA[RA2311027010032], TANISH SINGH[RA2311027010064]"** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Faculty in Charge
Dr. P. Rajasekar
Course Co-Ordinator
Department of DSBS
SRMIST - KTR

Head of Department
Dr. V. Kavitha
Prof & Head
Department of DSBS
SRMIST - KTR

ABSTRACT

This project is an **Attendance Management System** built using **Flask** and **SQLite** to streamline the process of tracking student attendance, managing academic records, and facilitating communication between students and teachers. The system is designed to provide a centralized and automated solution to replace traditional, paper-based attendance tracking and grading methods, improving efficiency and accuracy.

The database schema includes six key tables:

1. **Students** – Stores student details such as username, password, name, and email, ensuring secure authentication and access control.
2. **Teachers** – Stores teacher details, including login credentials and personal information, allowing teachers to manage student records and monitor attendance.
3. **Subjects** – Maintains a list of subjects with unique identifiers, providing a structured way to categorize academic data.
4. **Marks** – Records student marks for specific subjects and exam types, allowing teachers to input and update marks, and students to view their performance.
5. **Attendance** – Tracks student attendance status with links to both student and teacher records. It records details like login time and attendance date, providing accurate data for performance analysis.
6. **Sessions** – Manages user sessions, including session tokens and expiration times, to ensure secure login and prevent unauthorized access.

System Functionality

- **User Authentication** – Secure login for both students and teachers using encrypted passwords.
- **Attendance Tracking** – Real-time logging of attendance with timestamps and status.
- **Marks Management** – Teachers can input, update, and view student performance.
- **Data Security** – Passwords are encrypted, and session tokens ensure secure access.
- **Role-Based Access** – Students can only access personal data, while teachers can view and manage student records.
- **Performance Insights** – Data on attendance and academic performance can be analyzed to identify trends and improve student engagement.

Project Scope

The **Attendance Management System** aims to simplify and automate the process of tracking student attendance and managing academic records. The project scope defines the boundaries, features, and limitations of the system, ensuring a clear understanding of the objectives and deliverables. The system will provide a secure and efficient platform for managing student data, attendance records, and academic performance.

---

1. Objectives

- To automate the process of attendance tracking and reduce manual effort.
- To provide secure user authentication for both students and teachers.
- To allow teachers to update attendance records and marks efficiently.
- To enable students to check their attendance status and academic performance.
- To maintain accurate and secure data storage using SQLite.

---

2. Features User Management

- Secure login and session management for students and teachers.
- Password encryption and secure authentication.

Attendance Tracking

- Real-time logging of attendance records.
- Tracking login time and attendance status.
- Ability for teachers to update and monitor attendance.

Marks Management

- Input and update marks for various subjects and exam types.
- Allow students to view their marks and performance trends.

Data Management

- Centralized storage of student, teacher, subject, and performance data.
- Data integrity through relational database structure.

Role-Based Access Control

- Students can access their own attendance and marks.
- Teachers can manage and update student records.

Session Management

- Secure session tokens for authentication.
- Auto-expiration of sessions after a defined period.

3. Out of Scope

Integration with external systems (e.g., third-party databases, APIs).
Advanced data visualization or detailed performance analytics.
Biometric-based attendance tracking.
Offline functionality (system requires internet connectivity).

---

4. Stakeholders

- **Students** – To check attendance and academic performance.
- **Teachers** – To update attendance and marks.
- **Administrators** – To maintain the database and ensure system functionality.

---

5. Technical Scope

- **Backend:** Flask (Python-based) for handling business logic and user authentication.
- **Database:** SQLite for secure and fast data storage.
- **Frontend:** HTML, CSS, and JavaScript for the user interface.
- **Security:** Password encryption and secure session management using Flask libraries.

---

6. Success Criteria

Successful login and session management for students and teachers.
Accurate and secure storage of attendance and marks data.
User-friendly interface for both students and teachers.
Efficient performance with low latency and fast data retrieval.

---

7. Timeline

- **Phase 1:** Requirements gathering and database design.
- **Phase 2:** Backend and frontend development.
- **Phase 3:** Testing and debugging.
- **Phase 4:** Deployment and user feedback collection.

---

8. Conclusion

The project will provide a reliable and secure platform for managing student attendance and academic records. By automating manual processes, the system will save time, improve accuracy, and enhance the overall academic experience for both students and teachers.

Project Overview

The **Attendance Management System** is a web-based application developed using **Flask** (Python) and **SQLite** to automate the process of tracking student attendance and managing academic records. The system provides a centralized platform where teachers can monitor student attendance and performance, while students can view their attendance status and academic results in real-time. This project aims to enhance accuracy, reduce manual effort, and ensure secure access to data.

The main objectives of the project are to automate attendance tracking, provide secure login and session management for both students and teachers, allow teachers to update attendance and marks efficiently, enable students to view their attendance status and academic performance, and maintain secure and accurate data storage using SQLite.

Key features of the system include secure user authentication using encrypted passwords, real-time logging of attendance records with timestamps, marks management allowing teachers to input and update student marks, role-based access ensuring students can only view their data while teachers manage records, session management using tokens with expiration control, and data integrity through a relational database structure.

The system consists of six main components: the **Students** table stores student details such as username, password, and email; the **Teachers** table stores teacher details and login credentials; the **Subjects** table contains subject details and unique IDs; the **Marks** table records student marks for specific subjects and exams; the **Attendance** table logs student attendance status with timestamps; and the **Sessions** table manages user sessions and tokens.

The technology stack includes Flask for the backend, SQLite for data storage, and HTML, CSS, and JavaScript for the frontend. Security is ensured using Flask libraries for password encryption and session management.

The project scope includes attendance tracking and performance monitoring, secure user authentication and session management, and real-time data updates with role-based access. Out of scope are biometric attendance tracking, integration with external systems, and offline functionality.

The success of the project will be measured by successful login and session management, accurate recording of attendance and marks, a fast and responsive user interface, and secure and consistent data handling. The Attendance Management System will provide a secure and efficient platform for managing student attendance and academic records, improving accuracy and reducing the administrative workload for teachers.

TABLE OF CONTENTS

Problem understanding, Identification of Entity and Relationships, Construction of DB using ER Model for the Project

PROBLEM UNDERSTANDING

Challenges with Manual Attendance Tracking

Managing student attendance manually using physical registers or spreadsheets is highly inefficient and prone to errors. Teachers often have to take attendance through roll calls, which consumes valuable class time and increases the chances of mistakes. Proxy attendance is another common issue, where students mark attendance for absent classmates, leading to inaccurate records. Furthermore, teachers need to manually calculate attendance percentages, which is time-consuming and increases the likelihood of miscalculations. In cases where attendance records are maintained on paper or in Excel files, data loss or misplacement is a significant risk.

Difficulties in Academic Performance Tracking

Tracking student performance manually presents similar challenges. Teachers maintain student marks and academic records on paper or spreadsheets, which can be lost, corrupted, or accidentally deleted. Without a centralized system, it becomes difficult for teachers to retrieve historical performance data or analyze trends over time. Students often face delays in accessing their academic performance, leading to a lack of awareness about their progress and areas for improvement. Teachers also struggle to provide timely and targeted interventions because they lack real-time access to student performance data.

Security and Data Integrity Issues

Manual methods of attendance and performance tracking lack proper security measures, exposing sensitive student data to unauthorized access and manipulation. Without a structured system for user authentication and access control, it is difficult to protect confidential information such as student marks, attendance records, and personal details. Anyone with access to the records can modify them, leading to potential misuse or corruption of data. Session management is poorly handled in manual systems, making it difficult to maintain secure logins and prevent unauthorized access.

Lack of Real-Time Access and Reporting

Manual systems are not equipped to provide real-time access to attendance and academic performance data. Teachers often need to compile data from multiple sources to generate reports, which results in delays and inconsistencies. Students are unable to view their attendance status or academic performance in real time, limiting their ability to address issues promptly. Teachers also struggle to track patterns of absenteeism or poor performance, reducing their ability to provide timely support.

Absence of Role-Based Access Control

Without a clear differentiation between student and teacher roles, managing access to sensitive data becomes complex. In manual systems, students might inadvertently modify records, and teachers might lack proper access to historical data for analysis. A structured role-based access control system is essential to ensure that students can only access their data, while teachers can manage and update student records.

Need for an Automated Solution

To address these issues, an automated **Attendance Management System** is essential. By centralizing attendance and performance data, the system will eliminate the inefficiencies and inaccuracies of manual methods. Secure user authentication, real-time data access, and automated report generation will ensure that both students and teachers have the information they need to improve academic performance and attendance rates. The system will also provide structured session management, role-based access, and encrypted data storage to maintain the security and integrity of sensitive information. This project will solve the major challenges associated with manual attendance and performance tracking, improving the overall efficiency and accuracy of the academic management process.

IDENTIFICATION OF ENTITY AND RELATIONSHIPS

Entities and Relationships

The **Attendance Management System** includes the following key entities and relationships:

1. Students

- **Attributes:**
    - id – Primary Key (Unique identifier for each student)
    - username – Unique login name
    - password – Encrypted password for authentication
    - name – Full name of the student
    - email – Email address for communication
    - created_at – Timestamp for record creation
- **Relationships:**
    - One-to-Many with **Attendance** – A student can have multiple attendance records.

- One-to-Many with **Marks** – A student can have multiple marks records.
- One-to-Many with **Sessions** – A student can have multiple login sessions.

2. Teachers

- **Attributes:**
  - id – Primary Key (Unique identifier for each teacher)
  - name – Full name of the teacher
  - username – Unique login name
  - password – Encrypted password for authentication
  - email – Email address for communication
  - created_at – Timestamp for record creation
- **Relationships:**
  - One-to-Many with **Attendance** – A teacher can manage attendance records for multiple students.
  - No direct relationship with **Marks** (teachers manage the data but aren't linked directly).

3. Subjects

- **Attributes:**
  - id – Primary Key (Unique identifier for each subject)
  - sub_name – Name of the subject
- **Relationships:**
  - One-to-Many with **Marks** – A subject can have multiple marks entries for different students.

4. Marks

- **Attributes:**
    - id – Primary Key (Unique identifier for each marks record)
    - student_id – Foreign Key (References students.id)
    - subject_id – Foreign Key (References subjects.id)
    - marks – Marks obtained by the student
    - max_marks – Maximum possible marks for the exam
    - date – Date of the exam
    - exam_type – Type of exam (e.g., Midterm, Final)
- **Relationships:**
    - Many-to-One with **Students** – Each marks record is linked to a student.
    - Many-to-One with **Subjects** – Each marks record is linked to a subject.

5. Attendance

- **Attributes:**
    - id – Primary Key (Unique identifier for each attendance record)
    - student_id – Foreign Key (References students.id)
    - teacher_id – Foreign Key (References teachers.id)
    - date – Date of attendance
    - logged_in – Status indicating whether the student was present or absent
    - login_time – Timestamp of login or attendance mark
- **Relationships:**
    - Many-to-One with **Students** – Each attendance record is linked to a student.

- Many-to-One with **Teachers** – Each attendance record is linked to a teacher.

6. Sessions

- **Attributes:**
  - id – Primary Key (Unique identifier for each session)
  - student_id – Foreign Key (References students.id)
  - session_token – Unique token for session identification
  - created_at – Timestamp when the session was created
  - expires_at – Timestamp when the session will expire
- **Relationships:**
  - Many-to-One with **Students** – Each session is linked to a specific student.

Entity Relationships Overview

1. **Students** ↔ **Attendance** → One-to-Many (A student can have multiple attendance records).
2. **Students** ↔ **Marks** → One-to-Many (A student can have multiple marks records).
3. **Students** ↔ **Sessions** → One-to-Many (A student can have multiple login sessions).
4. **Teachers** ↔ **Attendance** → One-to-Many (A teacher can log attendance for multiple students).
5. **Subjects** ↔ **Marks** → One-to-Many (A subject can have multiple marks records).
6. **Students** ↔ **Subjects** → Many-to-Many (Through Marks).
7. **Teachers** ↔ **Students** → Indirect relationship through Attendance.

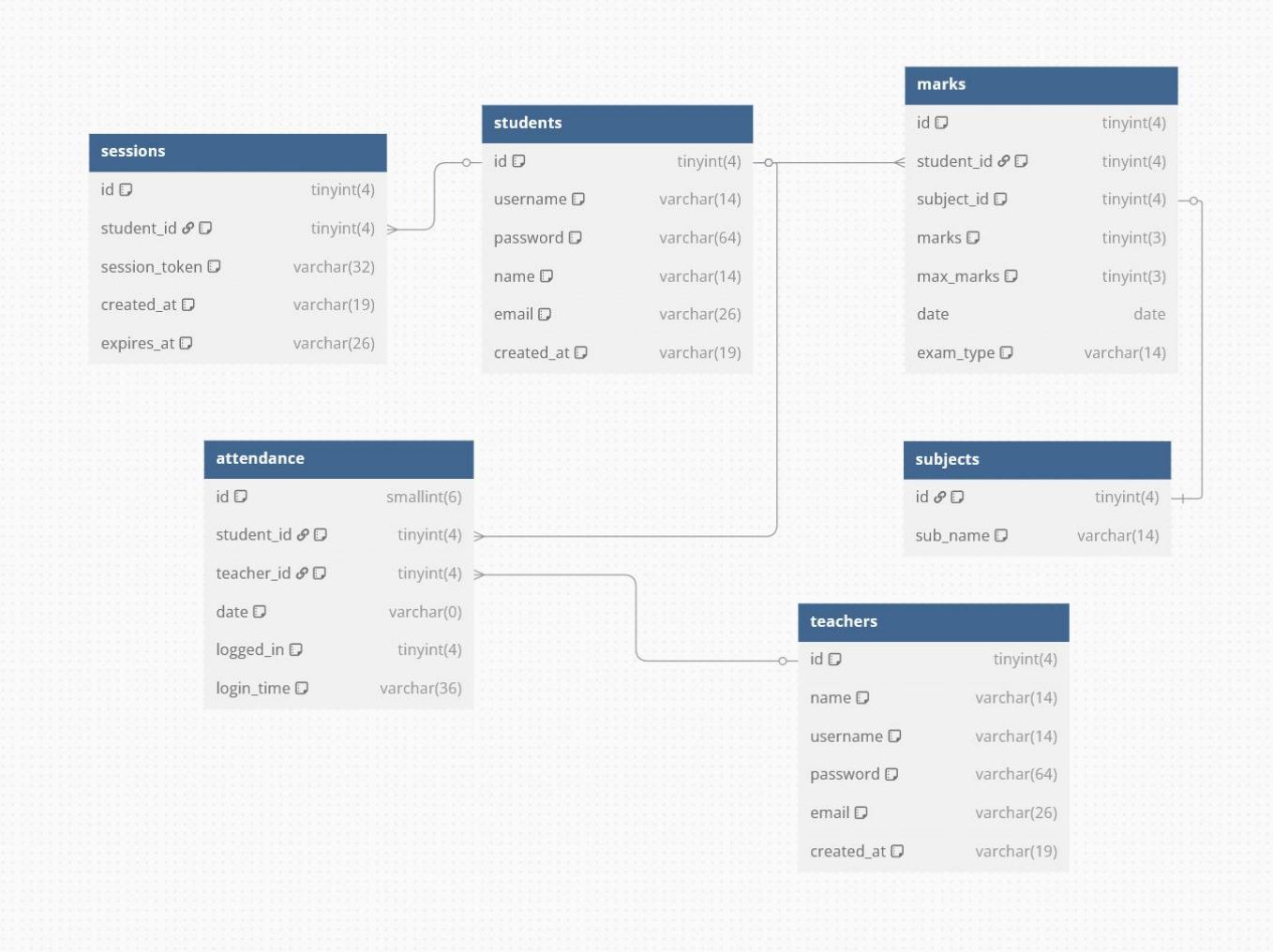
Summary

- The **Students** entity is central to the system, connecting to **Attendance**, **Marks**, and **Sessions**.
- **Teachers** are directly related to **Attendance** but not to **Marks**.

- **Subjects** are linked to **Marks** to keep track of student performance in different subjects.
- **Sessions** manage user authentication and secure access.
- The relationships ensure data consistency and integrity within the system.

Design of Relational Schema, Creation of Database Tables for the project

Relationship Schema



1. Student Table

Stores information about different students.

- **Attributes:**
  - id (Primary Key) → Unique ID for each student.
  - username → Unique login name.
  - password → Encrypted password for authentication.
  - name – Full name of the student
  - email – Email address for communication
  - created_at – Timestamp for record creation

- **Purpose:**
    - Each student can have multiple attendance records .

2. Teachers Table

Stores teachers details.

- **Attributes:**
    - id – Primary Key (Unique identifier for each teacher)
    - name – Full name of the teacher
    - username – Unique login name
    - password – Encrypted password for authentication
    - email – Email address for communication
    - created_at – Timestamp for record creation.
- **Purpose:**
    - A teacher can manage attendance records for multiple students.
    - No direct relationship with **Marks** (teachers manage the data but aren't linked directly).

3. Subjects Table

Stores information about different subjects.

- **Attributes:**
    - id – Primary Key (Unique identifier for each subject)
    - sub_name – Name of the subject
- **Purpose:**
    - A subject can have multiple marks entries for different students.

4. Show Table

Represents the shows available in theatres.

- **Attributes:**
    - show_id (Primary Key) → Unique ID for each show.
    - st_time → Show start time.
    - end_time → Show end time.
    - hall_no → Hall number within the theatre.
    - language → Language of the movie.
    - m_id (Foreign Key) → Refers to the **Movie** table.
- **Purpose:**
    - A show belongs to a **movie** and takes place in a **theatre**.
    - Connected to the **Tickets** table (each ticket belongs to a show).

5. Marks Table

Stores details of the marks of various students.

- **Attributes:**
  - id (Primary Key) → Primary Key (Unique identifier for each marks record).
  - m_name → Name of the movie.
  - student_id – Foreign Key (References students.id)
  - subject_id – Foreign Key (References subjects.id)
  - marks – Marks obtained by the student
  - max_marks – Maximum possible marks for the exam
  - date – Date of the exam
  - exam_type – Type of exam (e.g., Midterm, Final)
- **Purpose:**
  - Each marks record is linked to a student.
  - Each marks record is linked to a subject.

6. Attendance Table

Represents tickets issued for movie shows.

- **Attributes:**
  - id – Primary Key (Unique identifier for each attendance record)
  - student_id – Foreign Key (References students.id)
  - teacher_id – Foreign Key (References teachers.id)
  - date – Date of attendance
  - logged_in – Status indicating whether the student was present or absent
  - login_time – Timestamp of login or attendance mark
- **Purpose:**
  - Each attendance record is linked to a student.
  - Each attendance record is linked to a teacher.

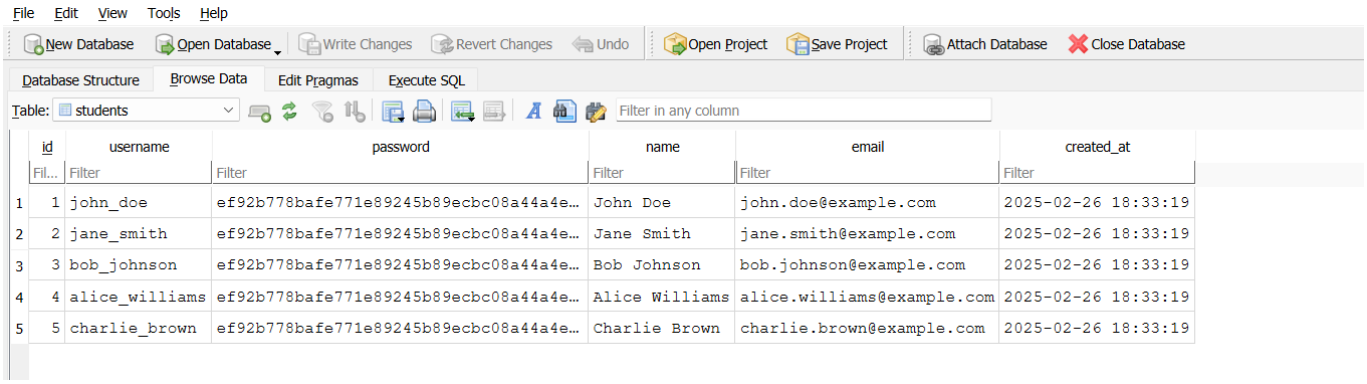
7. Sessions Table

Represents number of times the session has been logged into.

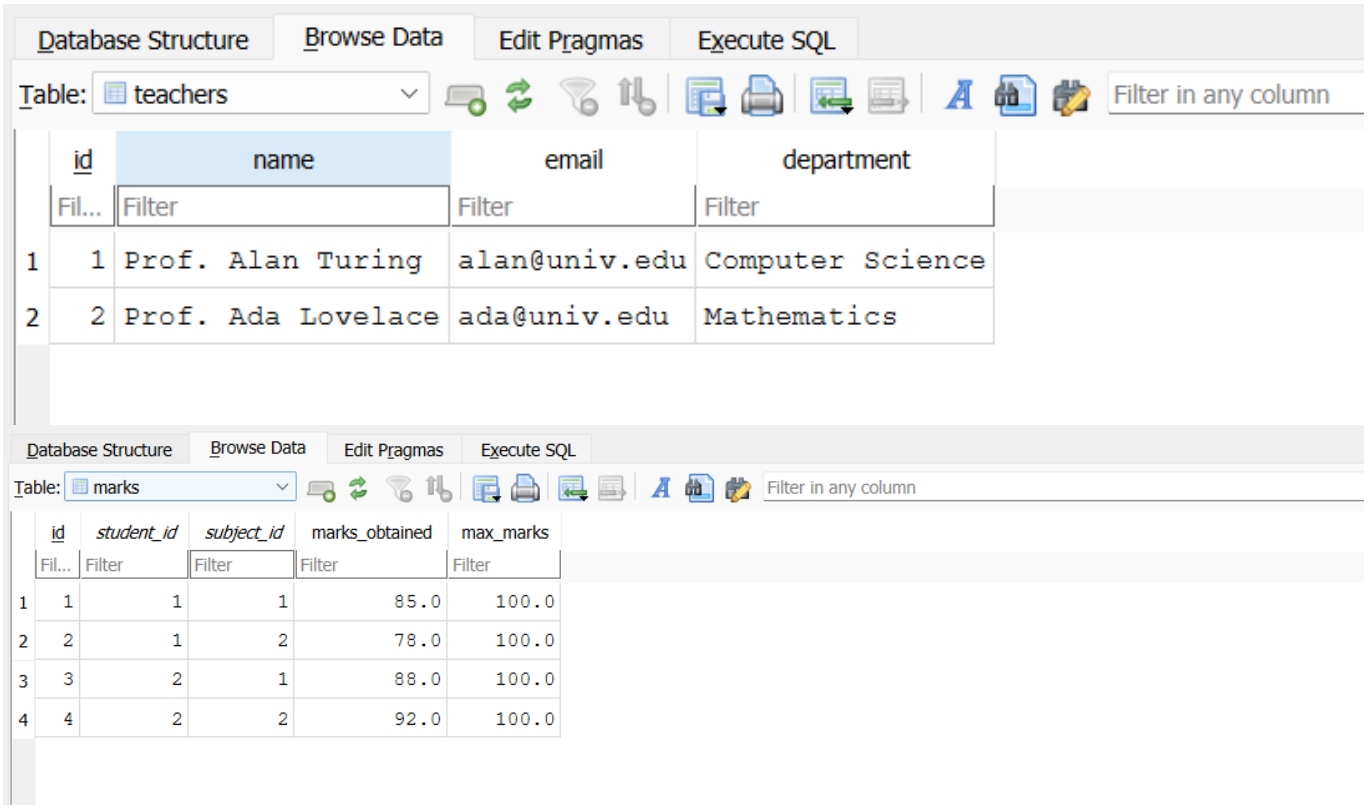
- **Attributes:**
  - id – Primary Key (Unique identifier for each session)
  - student_id – Foreign Key (References students.id)
  - session_token – Unique token for session identification
  - created_at – Timestamp when the session was created
  - expires_at – Timestamp when the session will expire

- **Purpose:**
  - Each session is linked to a specific student.

Database Tables

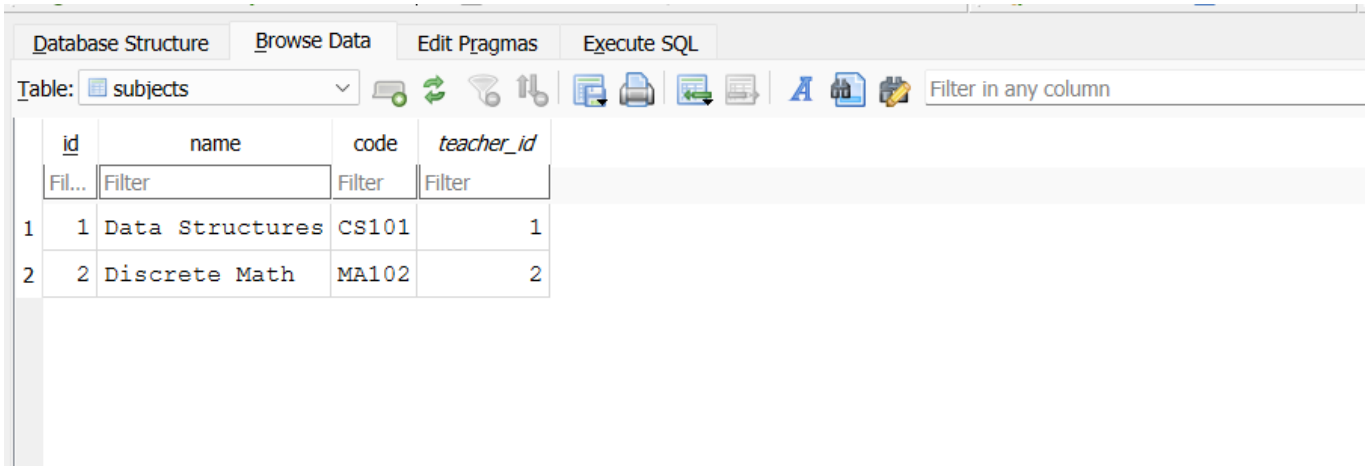


	id	username	password	name	email	created_at
1	1	john_doe	ef92b778bafe771e89245b89ecbc08a44a4e…	John Doe	john.doe@example.com	2025-02-26 18:33:19
2	2	jane_smith	ef92b778bafe771e89245b89ecbc08a44a4e…	Jane Smith	jane.smith@example.com	2025-02-26 18:33:19
3	3	bob_johnson	ef92b778bafe771e89245b89ecbc08a44a4e…	Bob Johnson	bob.johnson@example.com	2025-02-26 18:33:19
4	4	alice_williams	ef92b778bafe771e89245b89ecbc08a44a4e…	Alice Williams	alice.williams@example.com	2025-02-26 18:33:19
5	5	charlie_brown	ef92b778bafe771e89245b89ecbc08a44a4e…	Charlie Brown	charlie.brown@example.com	2025-02-26 18:33:19

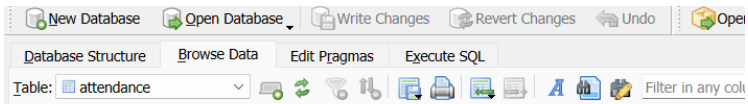


	id	name	email	department
1	1	Prof. Alan Turing	alan@univ.edu	Computer Science
2	2	Prof. Ada Lovelace	ada@univ.edu	Mathematics

	id	student_id	subject_id	marks_obtained	max_marks
1	1	1	1	85.0	100.0
2	2	1	2	78.0	100.0
3	3	2	1	88.0	100.0
4	4	2	2	92.0	100.0



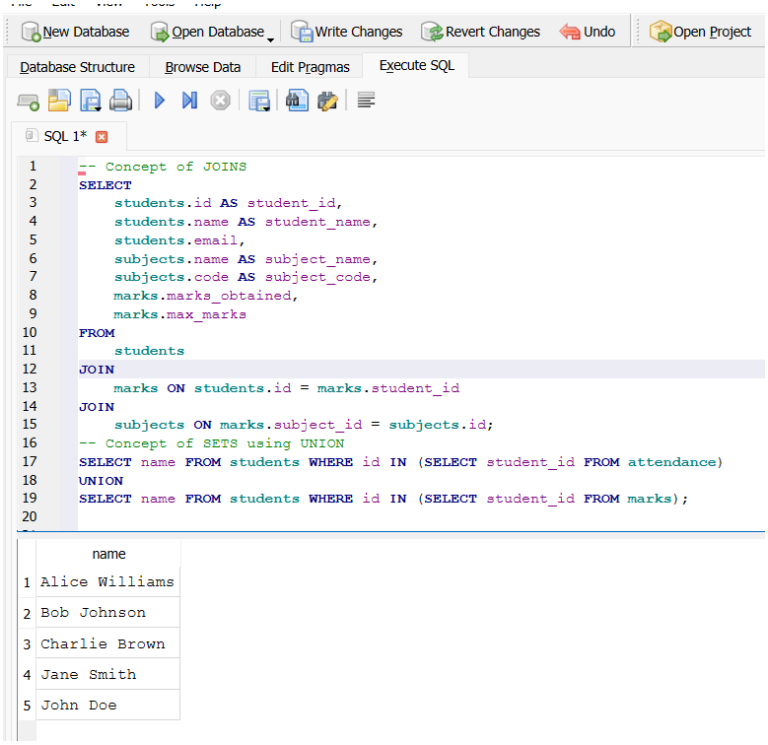
	id	name	code	teacher_id
	Fil...	Filter	Filter	Filter
1	1	Data Structures	CS101	1
2	2	Discrete Math	MA102	2

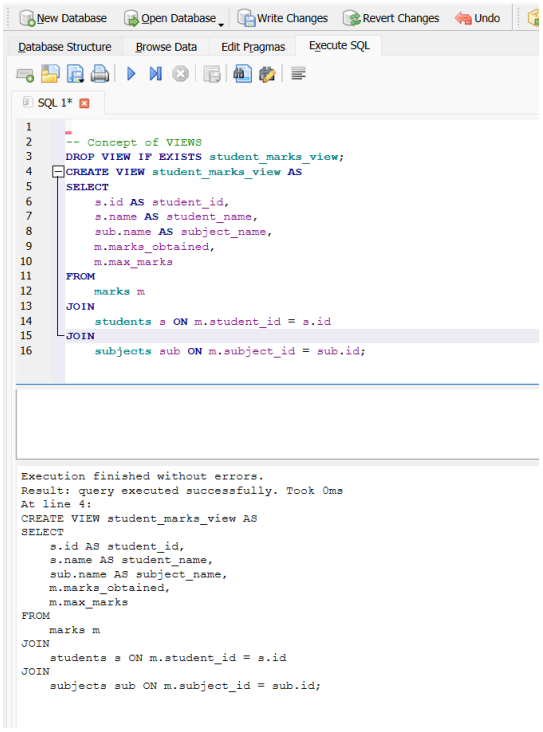


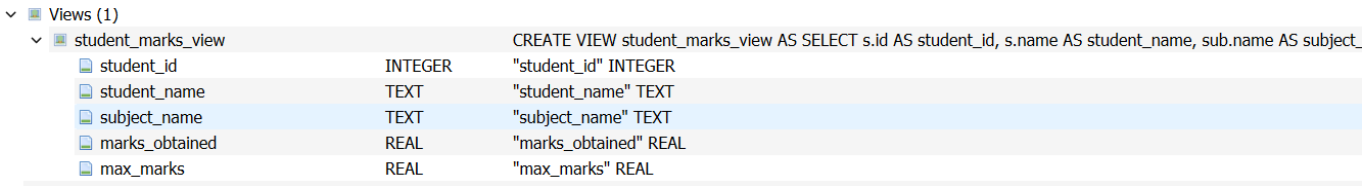
	id	student_id	date	logged_in	login_time
	Filter	Filter	Filter	Filter	Filter
1	1	1	2025-02-27	1	2025-02-27 18:13:15.179319
2	2	1	2025-02-26	1	2025-02-26 08:18:00
3	3	1	2025-02-25	1	2025-02-25 08:25:00
4	4	1	2025-02-24	1	2025-02-24 08:32:00
5	5	1	2025-02-23	1	2025-02-23 08:39:00
6	6	1	2025-02-22	1	2025-02-22 08:46:00
7	7	1	2025-02-21	1	2025-02-21 08:53:00
8	8	1	2025-02-20	0	NULL
9	9	1	2025-02-19	1	2025-02-19 08:07:00
10	10	1	2025-02-18	1	2025-02-18 08:14:00
11	11	1	2025-02-17	1	2025-02-17 08:21:00
12	12	1	2025-02-16	1	2025-02-16 08:28:00
13	13	1	2025-02-15	0	NULL
14	14	1	2025-02-14	1	2025-02-14 08:42:00
15	15	1	2025-02-13	0	NULL
16	16	1	2025-02-12	1	2025-02-12 08:56:00
17	17	1	2025-02-11	1	2025-02-11 08:03:00
18	18	1	2025-02-10	1	2025-02-10 08:10:00
19	19	1	2025-02-09	0	NULL

Complex Queries based on concept of constraints,sets,joins views,Triggers and Cursors

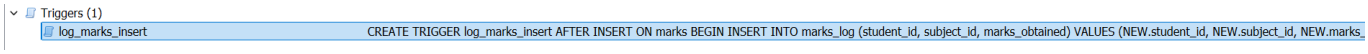
1.Concepts of joins ,sets and views

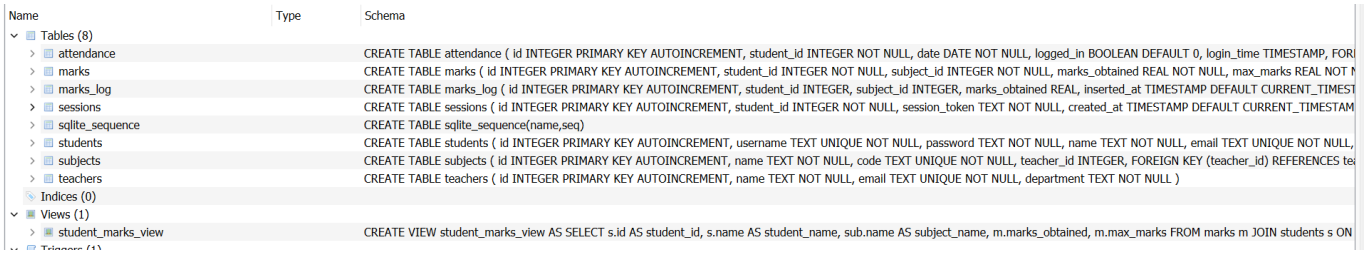






2.Concept of triggers and cursors





Name	Type	Schema
Tables (8)		
attendance		CREATE TABLE attendance (id INTEGER PRIMARY KEY AUTOINCREMENT, student_id INTEGER NOT NULL, date DATE NOT NULL, logged_in BOOLEAN DEFAULT 0, login_time TIMESTAMP, FOR
marks		CREATE TABLE marks (id INTEGER PRIMARY KEY AUTOINCREMENT, student_id INTEGER NOT NULL, subject_id INTEGER NOT NULL, marks_obtained REAL NOT NULL, max_marks REAL NOT N
marks_log		CREATE TABLE marks_log (id INTEGER PRIMARY KEY AUTOINCREMENT, student_id INTEGER, subject_id INTEGER, marks_obtained REAL, inserted_at TIMESTAMP DEFAULT CURRENT_TIMEST
sessions		CREATE TABLE sessions (id INTEGER PRIMARY KEY AUTOINCREMENT, student_id INTEGER NOT NULL, session_token TEXT NOT NULL, created_at TIMESTAMP DEFAULT CURRENT_TIMESTAM
sqlite_sequence		CREATE TABLE sqlite_sequence(name,seq)
students		CREATE TABLE students (id INTEGER PRIMARY KEY AUTOINCREMENT, username TEXT UNIQUE NOT NULL, password TEXT NOT NULL, name TEXT NOT NULL, email TEXT UNIQUE NOT NULL,
subjects		CREATE TABLE subjects (id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT NOT NULL, code TEXT UNIQUE NOT NULL, teacher_id INTEGER, FOREIGN KEY (teacher_id) REFERENCES te
teachers		CREATE TABLE teachers (id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT NOT NULL, email TEXT UNIQUE NOT NULL, department TEXT NOT NULL)
Indices (0)		
Views (1)		
student_marks_view		CREATE VIEW student_marks_view AS SELECT s.id AS student_id, s.name AS student_name, sub.name AS subject_name, m.marks_obtained, m.max_marks FROM marks m JOIN students s ON

```
    # --- Create Tables If Not Exist ---
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS students (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        username TEXT UNIQUE NOT NULL,
        password TEXT NOT NULL,
        name TEXT NOT NULL,
        email TEXT UNIQUE NOT NULL,
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    )
    ''')

    cursor.execute('''
    CREATE TABLE IF NOT EXISTS teachers (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL,
        email TEXT UNIQUE NOT NULL,
        password TEXT NOT NULL,
        department TEXT NOT NULL
    )
    ''')

    cursor.execute('''
    CREATE TABLE IF NOT EXISTS subjects (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL,
        code TEXT UNIQUE NOT NULL,
        teacher_id INTEGER,
        FOREIGN KEY (teacher_id) REFERENCES teachers (id)
    )
    ''')

    cursor.execute('''
```

Analyzing the Pitfalls, Dependencies,and applying normalization

1. Pitfalls in Your Current Design

a. Data Redundancy Risks

- If you store subject names in multiple places (e.g., marks, attendance), you risk inconsistencies if they are updated.
- Current design avoids this by linking via subject_id — that's good!

b. Lack of Role Separation in Authentication

- Students and teachers are authenticated differently (username vs. email), which can cause confusion and extra validation logic.
- **Suggestion:** Use a unified users table with role info (student/teacher) and shared fields like username, password.

c. No Soft Deletes

- If you delete a student or subject, associated marks and attendance may be lost due to ON DELETE not being handled.
- **Suggestion:** Add a soft delete mechanism (e.g., is_active flag).

d. No Indexing

- Querying large tables (e.g., for attendance over months) might be slow.
- **Suggestion:** Add indexes on foreign key columns like student_id, subject_id.

2. Dependencies in Your Project

Table	Depends on
marks	students, subjects
attendance	students
subjects	teachers
sessions	students
marks_log	marks (via trigger)

Implication:

- Changing student_id in students affects at least 3 tables (marks, attendance, sessions).
- Circular or overly coupled dependencies are not present — good!

---

3. Normalization in Your Schema

1NF (First Normal Form)

- ✓ Each table has atomic values (no arrays or lists in fields).
- ✓ Each row is uniquely identifiable (primary keys used properly).

2NF (Second Normal Form)

- ✓ All non-key attributes are fully functionally dependent on the primary key.
- E.g., in marks, marks_obtained depends on (student_id, subject_id) via composite uniqueness, though not enforced as a constraint.

3NF (Third Normal Form)

- ✓ No transitive dependency exists.
- E.g., student's email or name are not repeated in marks or attendance.

🗄️ 4. Normalize marks_log

If you're logging many other events, it may be worth having a logs table with columns:

```
(log_type, entity_id, description, created_at)
```

Instead of a separate marks_log.

```
CREATE TABLE attendance_sessions (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    attendance_day_id INTEGER NOT NULL,
    subject_id INTEGER, -- optional, to track subject
    login_time TIMESTAMP NOT NULL,
    logged_in BOOLEAN DEFAULT 1,
    FOREIGN KEY (attendance_day_id) REFERENCES attendance_days(id),
    FOREIGN KEY (subject_id) REFERENCES subjects(id)
);
```


```
CREATE TABLE attendance_sessions (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    attendance_day_id INTEGER NOT NULL,
    subject_id INTEGER, -- optional, to track subject
    login_time TIMESTAMP NOT NULL,
    logged_in BOOLEAN DEFAULT 1,
    FOREIGN KEY (attendance_day_id) REFERENCES attendance_days(id),
    FOREIGN KEY (subject_id) REFERENCES subjects(id)
);
```


Benefits of Normalization Here:

- Supports multiple logins per day (e.g., for different subjects).
- Tracks attendance more granularly.
- Avoids redundancy in date and student info.

Implementation of concurrency control and recovery mechanism

1. Concurrency Control

Problem:

SQLite allows multiple readers but only one writer at a time. Without proper handling, this can lead to **"database is locked"** errors.

Solutions:

a) Use BEGIN IMMEDIATE or BEGIN EXCLUSIVE for transactions:

Wrap critical write operations in a manual transaction to prevent partial updates.

```
python
CopyEdit
with get_db_connection() as conn:
    conn.execute("BEGIN IMMEDIATE")
    try:
        cursor = conn.cursor()
        # Perform multiple related DB operations
        cursor.execute("...")
        conn.commit()
    except Exception:
        conn.rollback()
        raise
```

b) Use a retry mechanism for locked DB:

Use Python logic to retry a few times if the DB is locked:

```
python
CopyEdit
import time

def safe_execute(query, params=()):
    attempts = 5
    for i in range(attempts):
        try:
            with get_db_connection() as conn:
                conn.execute(query, params)
                return
        except sqlite3.OperationalError as e:
            if "database is locked" in str(e).lower() and i < attempts - 1:
```

```
            time.sleep(0.5)
        else:
            raise
```

2. Recovery Mechanism

- **SQLite Built-in:**

  - **Automatic Rollback on Crash:** If using transactions (BEGIN ... COMMIT), SQLite will roll back on crash.
  - **WAL Mode (Write-Ahead Logging):** Provides atomic commit/rollback, better concurrency.

Enable WAL mode:

```
python
CopyEdit
with get_db_connection() as conn:
    conn.execute("PRAGMA journal_mode=WAL;")
```

3. Application-Level Logging for Recovery

Already using a trigger:

```
sql
CopyEdit
CREATE TRIGGER log_marks_insert ...
```

You can expand this with a **logs** table to store **manual logs** for key operations:

```
sql
CopyEdit
CREATE TABLE logs (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    operation TEXT NOT NULL,
    table_name TEXT NOT NULL,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    data TEXT
);
```

Log before critical operations (e.g., marks, attendance insert):

```
python
CopyEdit
cursor.execute("INSERT INTO logs (operation, table_name, data) VALUES (?, ?, ?)",
               ("INSERT", "marks", json.dumps({...})))
```

Summary

Mechanism	How to Implement
Concurrency Control	Use transactions, retry logic, and WAL mode
Recovery	Use triggers, transaction rollback, and custom logs
Backup	Export DB regularly using .backup command or scripts

Code for the Project

1.db_init.py

```
#!/usr/bin/env python3
"""
Database initialization script for attendance management system.
Creates tables if they don't exist and populates sample data if tables are empty.
"""

import sqlite3
import hashlib
import datetime
from config import DATABASE_PATH

# --- Password hashing ---
def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()

# --- Connect to DB ---
with sqlite3.connect(DATABASE_PATH) as conn:
    cursor = conn.cursor()



    cursor.execute('''
    CREATE TABLE IF NOT EXISTS teachers (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL,
        email TEXT UNIQUE NOT NULL,
        password TEXT NOT NULL,
        department TEXT NOT NULL
    )
    ''')

    cursor.execute('''
    CREATE TABLE IF NOT EXISTS subjects (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL,
        code TEXT UNIQUE NOT NULL,
        teacher_id INTEGER,
        FOREIGN KEY (teacher_id) REFERENCES teachers (id)
    )
    ''')
 # --- Insert Sample Data Only If Table is Empty ---
```

```
# Students
cursor.execute("SELECT COUNT(*) FROM students")
if cursor.fetchone()[0] == 0:
    students = [
        ('john_doe', hash_password('password123'), 'John Doe', 'john.doe@example.com'),
        ('jane_smith', hash_password('password123'), 'Jane Smith', 'jane.smith@example.com'),
        ('bob_johnson', hash_password('password123'), 'Bob Johnson', 'bob.johnson@example.com'),
        ('alice_williams', hash_password('password123'), 'Alice Williams', 'alice.williams@example.com'),
        ('charlie_brown', hash_password('password123'), 'Charlie Brown', 'charlie.brown@example.com'),
    ]
    cursor.executemany("INSERT INTO students (username, password, name, email) VALUES (?, ?, ?, ?)", students)

# Teachers
cursor.execute("SELECT COUNT(*) FROM teachers")
if cursor.fetchone()[0] == 0:
    teachers = [
        ('Prof. Alan Turing', 'alan@univ.edu', hash_password('turing123'), 'Computer Science'),
        ('Prof. Ada Lovelace', 'ada@univ.edu', hash_password('ada123'), 'Mathematics')
    ]
    cursor.executemany("INSERT INTO teachers (name, email, password, department) VALUES (?, ?, ?, ?)", teachers)

# Subjects
cursor.execute("SELECT COUNT(*) FROM subjects")
if cursor.fetchone()[0] == 0:
    subjects = [
        ('Data Structures', 'CS101', 1),
        ('Discrete Math', 'MA102', 2)
    ]
    cursor.executemany("INSERT INTO subjects (name, code, teacher_id) VALUES (?, ?, ?)", subjects)

# Marks
cursor.execute("SELECT COUNT(*) FROM marks")
if cursor.fetchone()[0] == 0:
    marks = [
        (1, 1, 85, 100),
        (1, 2, 78, 100),
        (2, 1, 88, 100),
        (2, 2, 92, 100)
    ]
    cursor.executemany("INSERT INTO marks (student_id, subject_id, marks_obtained, max_marks) VALUES (?, ?, ?, ?)", marks)
            )
```

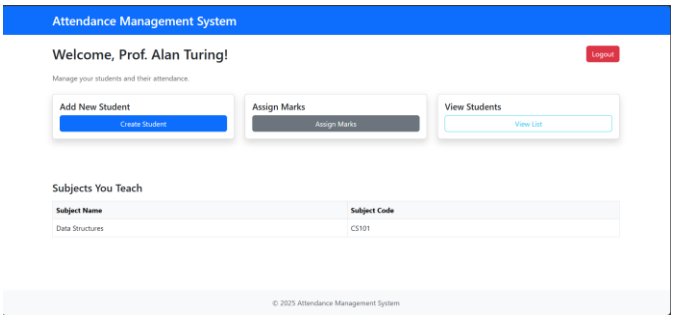
2.app.py

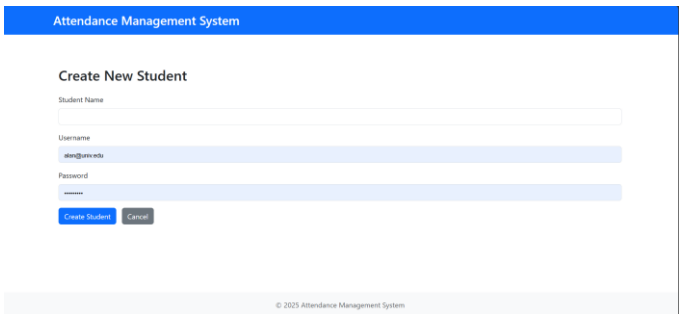
```python
from flask import Flask, render_template, redirect, url_for, request, session, flash
from models import Student, Teacher, Attendance, Session, Marks, Subject
from models import get_db_connection  # required for login auth
import hashlib
import traceback

app = Flask(__name__)
app.secret_key = "your_secret_key"  # Use a strong secret in production



# ---------- ROUTES ----------

@app.route('/')
def home():
    print("Root route '/' accessed.")
    role = session.get('role')
    if role:
        return redirect(url_for('dashboard'))
    return redirect(url_for('login'))



@app.route('/create_student', methods=['GET', 'POST'])
def create_student():
    if session.get('role') != 'teacher':
        flash("Unauthorized access.", "error")
        return redirect(url_for('login'))

    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('password')
        name = request.form.get('name')
        email = request.form.get('email')
        try:
            success = Teacher.create_student(username, password, name, email)
            if success:
                flash("Student created successfully!", "success")
            else:
                flash("Failed to create student. Username or email might already exist.", "error")
            return redirect(url_for('create_student'))
        except Exception as e:
            print("ERROR in create_student():", traceback.format_exc())
            flash(f'Error creating student: {str(e)}', 'error')

    return render_template('create_student.html')
```

```python
@app.route('/register')
def register():
    return "<h2>Registration not implemented yet.</h2>"



@app.route('/assign_marks', methods=['GET', 'POST'])
def assign_marks():
    if session.get('role') != 'teacher':
        flash("Unauthorized access.", "error")
        return redirect(url_for('login'))

    teacher_id = session.get('teacher_id')
    subjects = Subject.get_by_teacher(teacher_id)

    if request.method == 'POST':
        student_id = request.form.get('student_id')
        subject_id = request.form.get('subject_id')
        marks = request.form.get('marks')

        try:
            marks = float(marks)
            Teacher.assign_marks(student_id, subject_id, marks)
            flash("Marks assigned successfully.", "success")
        except Exception as e:
            flash("Failed to assign marks. Please check the details.", "error")
            print("Assign marks error:", e)

        return redirect(url_for('assign_marks'))

    with get_db_connection() as conn:
        students = conn.execute("SELECT id, name FROM students").fetchall()

    return render_template('assign_marks.html', students=students, subjects=subjects)
```

Result And Discussion

1.Teacher Dashboard



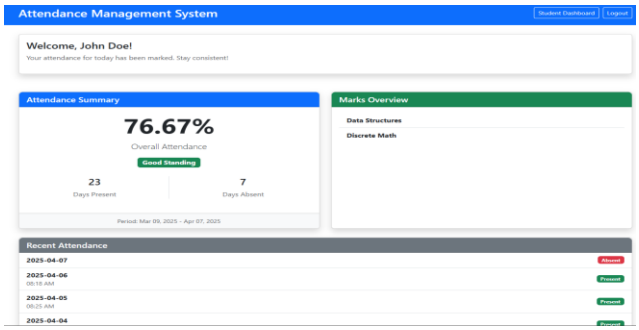


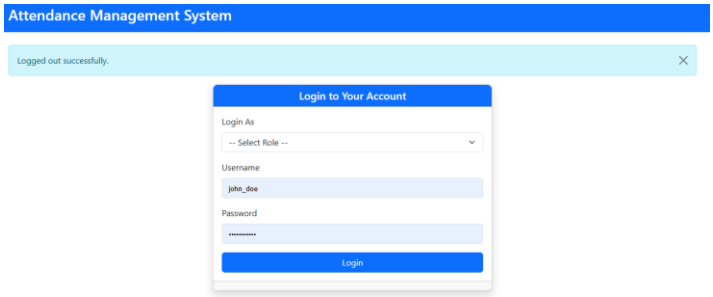
All Students

ID	Name	Email
1	John Doe	john.doe@example.com
2	Jane Smith	jane.smith@example.com
3	Bob Johnson	bob.johnson@example.com
4	Alice Williams	alice.williams@example.com
5	Charlie Brown	charlie.brown@example.com

Back to Dashboard

2.Student Dashboard





Conclusion

In conclusion, the **Attendance Management System** provides a comprehensive solution to the challenges faced in tracking student attendance and managing academic performance. By establishing a well-structured relational database, the system ensures accurate data storage and seamless access to attendance and marks records. The integration of secure user authentication, session management, and role-based access control enhances data security and prevents unauthorized access. The relationships between entities such as **Students**, **Teachers**, **Subjects**, **Marks**, **Attendance**, and **Sessions** create a cohesive structure that supports efficient data management and retrieval. This system not only reduces the administrative workload for teachers but also empowers students with real-time access to their academic progress, fostering a more transparent and accountable educational environment. Through automation and centralized data handling, the project improves overall efficiency, accuracy, and user experience in academic management.