Mid-Project Report: Library Management System

1. Project's Current Update

An overview of the completed tasks and implemented functionalities:

- **Database Setup**:
  - Successfully defined the User and Borrow_Record tables to manage user sign-ups and track book borrowing activities. The Borrow_Record table references the UserID to maintain a record of which user has borrowed a specific book.
  - Imported and cleaned Project Gutenberg's book database, replacing null values to ensure consistent data.
  - Inserted book records into the SQL database using MySQL.connector to allow the system to retrieve book information effectively.

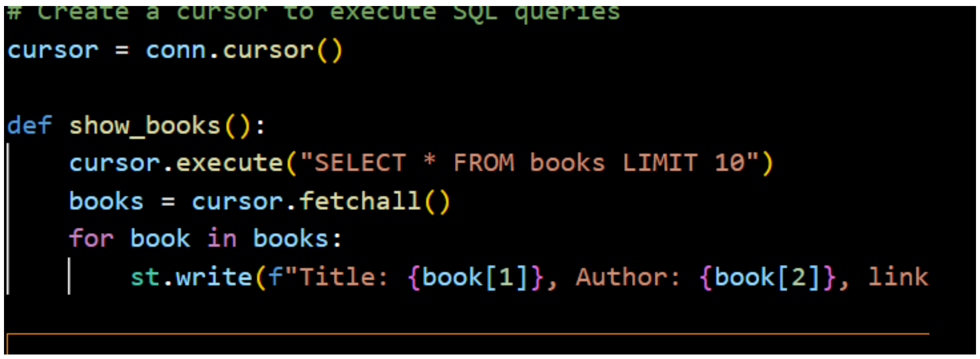
- **Core Functionalities Implemented**:
  - **Book Catalog System**: The system allows adding, editing, searching, and deleting book records for admins.
  - **User Account Management**: Users can register, and their borrowing history is maintained.
  - **Borrowing and Returns System**: Books can be checked out, and due dates are tracked.
  - **Search Functionality**: Users can search for books by title, author, or genre. This is accessible via the front-end interface.
- **Front-End Development**:
  - Began constructing the front end using Streamlit, which displays book titles and authors. This makes it easy for users to view available books.

2. Progress Compared to Milestone Set

- **Milestone 1**: Define the database schema (User, Borrow_Record, Book tables) and import book data (Completed).
- **Milestone 2**: Set up book catalog system (Add, Edit, Search, Delete) and user management system (Completed).
- **Milestone 3**: Implement borrowing and returns system (Completed).
- **Milestone 4**: Begin front-end development with Streamlit (Partially completed).
- The **AI recommendation system** is planned for future development, however, we are currently preparing the necessary structure (Not started).

3. Proof of the Project Update

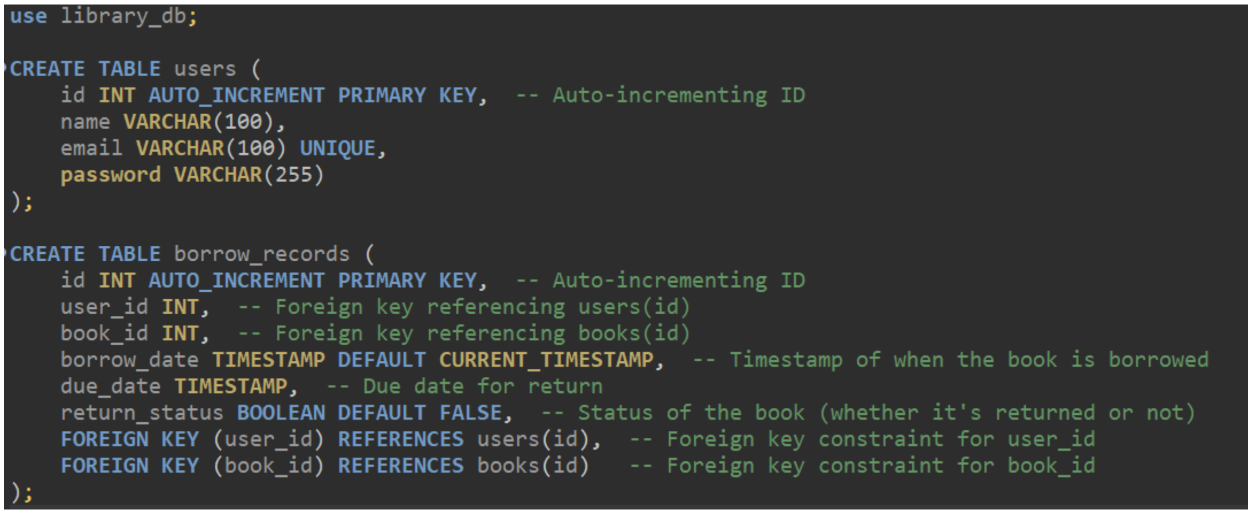
- **Screenshots of Front-End Interface code**: The current UI allows users to view book titles and authors.



```
# Create a cursor to execute SQL queries
cursor = conn.cursor()

def show_books():
    cursor.execute("SELECT * FROM books LIMIT 10")
    books = cursor.fetchall()
    for book in books:
        st.write(f"Title: {book[1]}, Author: {book[2]}, link
```

- **Database Screenshot**: A screenshot of the database tables, specifically the User and Borrow_Record tables.

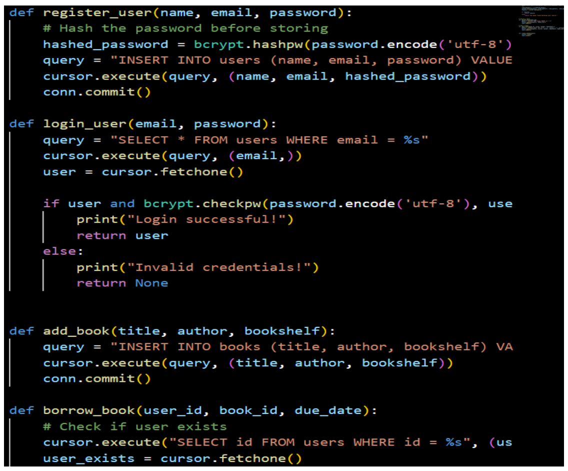


```
use library_db;

CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,  -- Auto-incrementing ID
    name VARCHAR(100),
    email VARCHAR(100) UNIQUE,
    password VARCHAR(255)
);

CREATE TABLE borrow_records (
    id INT AUTO_INCREMENT PRIMARY KEY,  -- Auto-incrementing ID
    user_id INT,  -- Foreign key referencing users(id)
    book_id INT,  -- Foreign key referencing books(id)
    borrow_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  -- Timestamp of when the book is borrowed
    due_date TIMESTAMP,  -- Due date for return
    return_status BOOLEAN DEFAULT FALSE,  -- Status of the book (whether it's returned or not)
    FOREIGN KEY (user_id) REFERENCES users(id),  -- Foreign key constraint for user_id
    FOREIGN KEY (book_id) REFERENCES books(id)   -- Foreign key constraint for book_id
);
```

- **Code Snippets**: Methods for registering users or adding books, user login and borrowing books.



```
def register_user(name, email, password):
    # Hash the password before storing
    hashed_password = bcrypt.hashpw(password.encode('utf-8')
    query = "INSERT INTO users (name, email, password) VALUE
    cursor.execute(query, (name, email, hashed_password))
    conn.commit()

def login_user(email, password):
    query = "SELECT * FROM users WHERE email = %s"
    cursor.execute(query, (email,))
    user = cursor.fetchone()

    if user and bcrypt.checkpw(password.encode('utf-8'), use
        print("Login successful!")
        return user
    else:
        print("Invalid credentials!")
        return None


def add_book(title, author, bookshelf):
    query = "INSERT INTO books (title, author, bookshelf) VA
    cursor.execute(query, (title, author, bookshelf))
    conn.commit()

def borrow_book(user_id, book_id, due_date):
    # Check if user exists
    cursor.execute("SELECT id FROM users WHERE id = %s", (us
    user_exists = cursor.fetchone()
```

4. Future Work

- **Complete Front-End Development**: Enhance the front-end with additional features such as displaying more book details, as well as allowing users to borrow and return books through the UI.
- **AI Recommendation System**: Implement the AI system that will recommend books to users based on their browsing history and borrowing patterns.
- **User Interface Improvements and Testing**: Make the UI more interactive, work on optimizing search functionalities, and perform testing to identify and fix bugs, ensuring smooth functionality across all systems.