ANURAG UNIVERSITY | SCHOOL OF ENGINEERING

DEPARTMENT OF ARTIFICIAL INTELLIGENCE

Project Based Report on

Online Book Store

TEAM 4

23 EG 106 E08

23 EG 106 E15

23 EG 106 E16

23 EG 106 E17

Presentation Schedule :

Programme : B.Tech (2024-25)

Year/ Semester : 2024-24

Class & Section : AI –E Section

Course Name : Data Base Management Systems

Faculty Name : S.Vijitha

Online Book Store

Overview:

This web application is a Bookstore Management System developed using Flask and SQLite. It allows users to browse books, add them to their cart, and place orders. Admins can manage books, categories, and view orders.

TechStack:

- Backend: Flask (Python)
- Database: SQLite (database.sqlite3)
- Frontend: HTML, CSS (Jinja Templates)
- Authentication: Session-based login system

Database Schema & Relationships:

Table Name	Description
`authors`	Stores author details (name, bio).
`books`	Stores book details (title, author, price, stock, etc.).
`categories`	Stores book categories.
`books_categories`	A **many-to-many relationship** table linking books to categories.
`users`	Stores user details, including authentication info and admin status.
`cart`	Stores books added to the user's shopping cart.
`orders`	Stores placed orders with total amount and shipping address.
`order_items`	Stores items within each order, linking to books.

Key Relationships:

- One-to-Many:
- books → authors (Each book is written by one author)

- orders → users (Each order belongs to a single user)
- cart → users (Each cart entry is linked to a user)
- order_items → orders (Each item belongs to one order)
- order_items → books (Each order item refers to a book)
- Many-to-Many:
- books ↔ categories (A book can belong to multiple categories, and each category can have multiple books)

Feature Breakdown: Authentication & Order Processing:

1. Authentication System (User Login & Session Management):

The application handles user authentication using Flask sessions.

How It Works:

1. **User Registration:**
   - Users sign up with username, email, and password.
   - Passwords are stored unencrypted in SQLite (⚠ should use hashing like bcrypt for security).
2. **User Login:**
   - The app verifies username and password from the users table.
   - If credentials match, a Flask session is created.
   - Users are redirected to their dashboard.
3. **Session Management:**
   - Flask sessions store user authentication details.
   - Users remain logged in until they log out or the session expires.
4. **Admin Access:**
   - The is_admin column in the users table determines admin rights.
   - Admins get extra privileges like managing books, authors, and orders.

2. Order Processing System

How It Works:

1. **Adding Items to Cart**:
   - Users add books to their **cart**, stored in the cart table.
   - Each cart entry has user_id, book_id, and quantity.

2. **Placing an Order**:
   - When a user proceeds to checkout:
     - A new entry is created in the orders table.
     - The total amount is calculated from cart items.
     - Shipping details are stored.
3. **Order Items Storage**:
   - Each book in the cart is moved to order_items, linking it to the order.
   - The book stock is updated (reduced by the purchased quantity).
4. **Order Status Management**:
   - Orders have a **status** (Pending, Shipped, Delivered).
   - Admins can update the order status from the dashboard.

3. Code Snippets for Key Functionalities:

```
from flask import Flask, request, jsonify, render_template, redirect, url_for, session, Response

import sqlite3

import os


app = Flask(__name__)

app.secret_key = 'bookworm_haven_secret_key'


# Database initialization

def get_db_connection():

    conn = sqlite3.connect('database.sqlite3')

    conn.row_factory = sqlite3.Row

    return conn


def table_exists(conn, table_name):

    """Check if a table exists in the database"""

    cursor = conn.cursor()

    cursor.execute(f"SELECT name FROM sqlite_master WHERE type='table' AND
name='{table_name}'")

    return cursor.fetchone() is not None


def init_db():
```

```
"""Initialize database tables if they don't exist"""
conn = get_db_connection()
cursor = conn.cursor()

# Create Authors table if it doesn't exist
if not table_exists(conn, 'authors'):
    cursor.execute('''
    CREATE TABLE authors (
        author_id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL,
        bio TEXT
    )
    ''')

# Create Books table if it doesn't exist
if not table_exists(conn, 'books'):
    cursor.execute('''
    CREATE TABLE books (
        book_id INTEGER PRIMARY KEY,
        title TEXT NOT NULL,
        author_id INTEGER NOT NULL,
        stock INTEGER NOT NULL,
        price REAL NOT NULL,
        published_date TEXT NOT NULL,
        information TEXT,
        image_path TEXT,
        FOREIGN KEY (author_id) REFERENCES authors (author_id)
    )
    ''')
```

```
    # Create Categories table if it doesn't exist
    if not table_exists(conn, 'categories'):
        cursor.execute('''
        CREATE TABLE categories (
            category_id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT NOT NULL
        )
        ''')

        # Insert default categories
        categories = ['Fiction', 'Mystery', 'Science Fiction', 'Self Help', 'Biography', 'Romance',
'History']
        for category in categories:
            cursor.execute('INSERT INTO categories (name) VALUES (?)', (category,))

    # Create Books_Categories junction table if it doesn't exist
    if not table_exists(conn, 'books_categories'):
        cursor.execute('''
        CREATE TABLE books_categories (
            book_id INTEGER,
            category_id INTEGER,
            PRIMARY KEY (book_id, category_id),
            FOREIGN KEY (book_id) REFERENCES books (book_id),
            FOREIGN KEY (category_id) REFERENCES categories (category_id)
        )
        ''')

    # Create Users table if it doesn't exist
```

```
    if not table_exists(conn, 'users'):
        cursor.execute('''
        CREATE TABLE users (
            user_id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT UNIQUE NOT NULL,
            password TEXT NOT NULL,
            email TEXT UNIQUE NOT NULL,
            is_admin BOOLEAN DEFAULT 0
        )
        ''')

        # Insert default admin user
        cursor.execute('INSERT INTO users (username, password, email, is_admin) VALUES (?, ?, ?, ?)',
                       ('admin', 'admin123', 'admin@bookworm.com', 1))

    # Create Cart table if it doesn't exist
    if not table_exists(conn, 'cart'):
        cursor.execute('''
        CREATE TABLE cart (
            cart_id INTEGER PRIMARY KEY AUTOINCREMENT,
            user_id INTEGER NOT NULL,
            book_id INTEGER NOT NULL,
            quantity INTEGER NOT NULL,
            FOREIGN KEY (user_id) REFERENCES users (user_id),
            FOREIGN KEY (book_id) REFERENCES books (book_id)
        )
        ''')
```

```
# Create Orders table if it doesn't exist
if not table_exists(conn, 'orders'):
    cursor.execute('''
    CREATE TABLE orders (
        order_id INTEGER PRIMARY KEY AUTOINCREMENT,
        user_id INTEGER NOT NULL,
        order_date TEXT NOT NULL,
        status TEXT NOT NULL,
        total_amount REAL NOT NULL,
        shipping_address TEXT NOT NULL,
        FOREIGN KEY (user_id) REFERENCES users (user_id)
    )
    ''')

# Create Order_Items table if it doesn't exist
if not table_exists(conn, 'order_items'):
    cursor.execute('''
    CREATE TABLE order_items (
        order_item_id INTEGER PRIMARY KEY AUTOINCREMENT,
        order_id INTEGER NOT NULL,
        book_id INTEGER NOT NULL,
        quantity INTEGER NOT NULL,
        price REAL NOT NULL,
        FOREIGN KEY (order_id) REFERENCES orders (order_id),
        FOREIGN KEY (book_id) REFERENCES books (book_id)
    )
    ''')

conn.commit()
```

```
    conn.close()

# Initialize database tables
init_db()

# Routes
@app.route("/")
def index():
    conn = get_db_connection()
    # Get featured books (latest 4 books)
    featured_books = conn.execute('''
    SELECT b.book_id, b.title, b.price, b.image_path, a.name as author_name
    FROM books b
    JOIN authors a ON b.author_id = a.author_id
    ORDER BY b.book_id DESC LIMIT 4
    ''').fetchall()
    conn.close()

    return render_template("index.html", featured_books=featured_books)

@app.route("/books")
def books():
    search = request.args.get('search', '')
    category = request.args.get('category', '')
    sort = request.args.get('sort', 'newest')

    conn = get_db_connection()

    # Base query
    query = '''
```

```
    SELECT b.book_id, b.title, b.price, b.stock, b.information, b.image_path,
           a.name as author_name, b.published_date
    FROM books b
    JOIN authors a ON b.author_id = a.author_id
    '''

    params = []

    # Add search filter if provided
    if search:
        query += ' WHERE b.title LIKE ? OR a.name LIKE ?'
        search_param = f'%{search}%'
        params.extend([search_param, search_param])

    # Add category filter if provided
    if category:
        if 'WHERE' in query:
            query += ' AND b.book_id IN (SELECT book_id FROM books_categories WHERE
category_id = ?)'
        else:
            query += ' WHERE b.book_id IN (SELECT book_id FROM books_categories
WHERE category_id = ?)'
        params.append(category)

    # Add sorting
    if sort == 'price-low':
        query += ' ORDER BY b.price ASC'
    elif sort == 'price-high':
        query += ' ORDER BY b.price DESC'
```

```
    elif sort == 'newest':
        query += ' ORDER BY b.published_date DESC'
    else:  # Default to newest
        query += ' ORDER BY b.published_date DESC'

    books = conn.execute(query, params).fetchall()

    # Get all categories for filter dropdown
    categories = conn.execute('SELECT * FROM categories').fetchall()

    conn.close()

    return render_template("books.html", books=books, categories=categories,
                           search=search, selected_category=category, selected_sort=sort)

@app.route("/book/<int:book_id>")
def book_detail(book_id):
    conn = get_db_connection()
    book = conn.execute('''
    SELECT b.*, a.name as author_name, a.bio as author_bio
    FROM books b
    JOIN authors a ON b.author_id = a.author_id
    WHERE b.book_id = ?
    ''', (book_id,)).fetchone()

    if not book:
        conn.close()
        return render_template("error.html", message="Book not found"), 404
```

```
    # Get book categories
    book_categories = conn.execute('''
    SELECT c.name
    FROM categories c
    JOIN books_categories bc ON c.category_id = bc.category_id
    WHERE bc.book_id = ?
    ''', (book_id,)).fetchall()

    conn.close()

    return render_template("book_detail.html", book=book, categories=book_categories)

@app.route("/authors")
def authors():
    conn = get_db_connection()
    authors = conn.execute('SELECT * FROM authors').fetchall()
    conn.close()

    return render_template("authors.html", authors=authors)

@app.route("/author/<int:author_id>")
def author_detail(author_id):
    conn = get_db_connection()
    author = conn.execute('SELECT * FROM authors WHERE author_id = ?',
(author_id,)).fetchone()

    if not author:
        conn.close()
        return render_template("error.html", message="Author not found"), 404
```

```
    # Get author's books
    author_books = conn.execute('''
    SELECT book_id, title, price, published_date, image_path
    FROM books
    WHERE author_id = ?
    ORDER BY published_date DESC
    ''', (author_id,)).fetchall()

    conn.close()

    return render_template("author_detail.html", author=author, books=author_books)

@app.route("/add_author", methods=['GET', 'POST'])
def add_author():
    # Check if user is logged in
    if 'user_id' not in session:
        return redirect(url_for('login', next=url_for('add_author')))

    if request.method == 'POST':
        name = request.form.get('name')
        bio = request.form.get('bio', '')

        if not name:
            return render_template("add_author.html", error="Author name is required")

        conn = get_db_connection()
        conn.execute('INSERT INTO authors (name, bio) VALUES (?, ?)', (name, bio))
        conn.commit()
        conn.close()
```

```
        return redirect(url_for('authors'))

    return render_template("add_author.html")

@app.route("/sell", methods=['GET', 'POST'])
def sell_book():
    # Check if user is logged in
    if 'user_id' not in session:
        return redirect(url_for('login', next=url_for('sell_book')))

    if request.method == 'POST':
        book_id = request.form.get('book_id')
        title = request.form.get('title')
        author_id = request.form.get('author_id')
        stock = request.form.get('stock')
        price = request.form.get('price')
        published_date = request.form.get('published_date')
        information = request.form.get('information', '')

        # Validate required fields
        if not all([book_id, title, author_id, stock, price, published_date]):
            conn = get_db_connection()
            authors = conn.execute('SELECT * FROM authors').fetchall()
            conn.close()
            return render_template("sell.html", authors=authors,
                                   error="All fields except Information are required")

        # Validate book_id is unique
```

```
    conn = get_db_connection()
    existing_book = conn.execute('SELECT * FROM books WHERE book_id = ?',
                                 (book_id,)).fetchone()

    if existing_book:
        authors = conn.execute('SELECT * FROM authors').fetchall()
        conn.close()
        return render_template("sell.html", authors=authors,
                               error="A book with this ID already exists")

    # Insert the new book
    conn.execute('''
    INSERT INTO books (book_id, title, author_id, stock, price, published_date,
information)
    VALUES (?, ?, ?, ?, ?, ?, ?)
    ''', (book_id, title, author_id, stock, price, published_date, information))

    # Handle category assignments
    categories = request.form.getlist('categories')
    for category_id in categories:
        conn.execute('INSERT INTO books_categories (book_id, category_id) VALUES (?, ?)',
                     (book_id, category_id))

    conn.commit()
    conn.close()

    return redirect(url_for('books'))

conn = get_db_connection()
```

```
    authors = conn.execute('SELECT * FROM authors').fetchall()
    categories = conn.execute('SELECT * FROM categories').fetchall()
    conn.close()

    return render_template("sell.html", authors=authors, categories=categories)

@app.route("/add_to_cart", methods=['POST'])
def add_to_cart():
    if 'user_id' not in session:
        return redirect(url_for('login', next=request.referrer))

    book_id = request.form.get('book_id')
    quantity = int(request.form.get('quantity', 1))

    conn = get_db_connection()

    # Check if book exists and has enough stock
    book = conn.execute('SELECT stock FROM books WHERE book_id = ?',
(book_id,)).fetchone()

    if not book:
        conn.close()
        return render_template("error.html", message="Book not found"), 404

    if book['stock'] < quantity:
        conn.close()
        return render_template("error.html", message="Not enough books in stock"), 400

    # Check if book is already in cart
```

```
    existing_item = conn.execute('''
    SELECT cart_id, quantity FROM cart
    WHERE user_id = ? AND book_id = ?
    ''', (session['user_id'], book_id)).fetchone()

    if existing_item:
        # Update quantity
        new_quantity = existing_item['quantity'] + quantity
        conn.execute('UPDATE cart SET quantity = ? WHERE cart_id = ?',
                     (new_quantity, existing_item['cart_id']))
    else:
        # Add new cart item
        conn.execute('''
        INSERT INTO cart (user_id, book_id, quantity)
        VALUES (?, ?, ?)
        ''', (session['user_id'], book_id, quantity))

    conn.commit()
    conn.close()

    return redirect(url_for('cart'))

@app.route("/cart")
def cart():
    if 'user_id' not in session:
        return redirect(url_for('login', next=request.url))

    conn = get_db_connection()
    cart_items = conn.execute('''
```

```
    SELECT c.cart_id, c.quantity, b.book_id, b.title, b.price, b.image_path, a.name as
author_name
    FROM cart c
    JOIN books b ON c.book_id = b.book_id
    JOIN authors a ON b.author_id = a.author_id
    WHERE c.user_id = ?
    ''', (session['user_id'],)).fetchall()

    # Calculate total
    total = 0
    for item in cart_items:
        total += item['price'] * item['quantity']

    conn.close()

    return render_template("cart.html", cart_items=cart_items, total=total)

@app.route("/update_cart", methods=['POST'])
def update_cart():
    if 'user_id' not in session:
        return redirect(url_for('login'))

    cart_id = request.form.get('cart_id')
    quantity = int(request.form.get('quantity', 0))

    conn = get_db_connection()

    if quantity <= 0:
        # Remove item from cart
```

```
        conn.execute('DELETE FROM cart WHERE cart_id = ? AND user_id = ?',
                     (cart_id, session['user_id']))
    else:
        # Check stock availability
        cart_item = conn.execute('''
        SELECT b.stock, c.book_id
        FROM cart c
        JOIN books b ON c.book_id = b.book_id
        WHERE c.cart_id = ? AND c.user_id = ?
        ''', (cart_id, session['user_id'])).fetchone()

        if not cart_item:
            conn.close()
            return render_template("error.html", message="Cart item not found"), 404

        if cart_item['stock'] < quantity:
            conn.close()
            return render_template("error.html", message="Not enough books in stock"), 400

        # Update quantity
        conn.execute('UPDATE cart SET quantity = ? WHERE cart_id = ? AND user_id = ?',
                     (quantity, cart_id, session['user_id']))

    conn.commit()
    conn.close()

    return redirect(url_for('cart'))

@app.route("/checkout", methods=['GET', 'POST'])
```

```
def checkout():
    if 'user_id' not in session:
        return redirect(url_for('login', next=url_for('checkout')))

    if request.method == 'POST':
        shipping_address = request.form.get('shipping_address')

        if not shipping_address:
            return render_template("checkout.html", error="Shipping address is required")

        conn = get_db_connection()

        # Get cart items
        cart_items = conn.execute('''
        SELECT c.book_id, c.quantity, b.price, b.stock
        FROM cart c
        JOIN books b ON c.book_id = b.book_id
        WHERE c.user_id = ?
        ''', (session['user_id'],)).fetchall()

        if not cart_items:
            conn.close()
            return render_template("error.html", message="Your cart is empty"), 400

        # Check stock availability and calculate total
        total_amount = 0
        for item in cart_items:
            if item['stock'] < item['quantity']:
                conn.close()
```

```
        return render_template("error.html",
                               message=f"Not enough stock for book ID {item['book_id']}"), 400
    total_amount += item['price'] * item['quantity']

# Create order
import datetime
order_date = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')
cursor = conn.cursor()
cursor.execute('''
INSERT INTO orders (user_id, order_date, status, total_amount, shipping_address)
VALUES (?, ?, ?, ?, ?)
''', (session['user_id'], order_date, 'Pending', total_amount, shipping_address))

order_id = cursor.lastrowid

# Add order items and update stock
for item in cart_items:
    # Add order item
    conn.execute('''
    INSERT INTO order_items (order_id, book_id, quantity, price)
    VALUES (?, ?, ?, ?)
    ''', (order_id, item['book_id'], item['quantity'], item['price']))

    # Update stock
    new_stock = item['stock'] - item['quantity']
    conn.execute('UPDATE books SET stock = ? WHERE book_id = ?',
                 (new_stock, item['book_id']))

# Clear cart
```

```
        conn.execute('DELETE FROM cart WHERE user_id = ?', (session['user_id'],))

        conn.commit()
        conn.close()

        return redirect(url_for('order_confirmation', order_id=order_id))

    conn = get_db_connection()
    cart_items = conn.execute('''
    SELECT c.quantity, b.title, b.price
    FROM cart c
    JOIN books b ON c.book_id = b.book_id
    WHERE c.user_id = ?
    ''', (session['user_id'],)).fetchall()

    # Calculate total
    total = 0
    for item in cart_items:
        total += item['price'] * item['quantity']

    conn.close()

    return render_template("checkout.html", cart_items=cart_items, total=total)

@app.route("/order_confirmation/<int:order_id>")
def order_confirmation(order_id):
    if 'user_id' not in session:
        return redirect(url_for('login'))
```

```
    conn = get_db_connection()
    order = conn.execute('''
    SELECT * FROM orders WHERE order_id = ? AND user_id = ?
    ''', (order_id, session['user_id'])).fetchone()

    if not order:
        conn.close()
        return render_template("error.html", message="Order not found"), 404

    order_items = conn.execute('''
    SELECT oi.quantity, oi.price, b.title
    FROM order_items oi
    JOIN books b ON oi.book_id = b.book_id
    WHERE oi.order_id = ?
    ''', (order_id,)).fetchall()

    conn.close()

    return render_template("order_confirmation.html", order=order, items=order_items)

@app.route("/orders")
def order_history():
    if 'user_id' not in session:
        return redirect(url_for('login', next=url_for('order_history')))

    conn = get_db_connection()
    orders = conn.execute('''
    SELECT order_id, order_date, status, total_amount
    FROM orders
```

```
    WHERE user_id = ?
    ORDER BY order_date DESC
    ''', (session['user_id'],)).fetchall()

    conn.close()

    return render_template("orders.html", orders=orders)

@app.route("/login", methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('password')

        conn = get_db_connection()
        user = conn.execute('''
        SELECT * FROM users WHERE username = ? AND password = ?
        ''', (username, password)).fetchone()

        if user:
            session['user_id'] = user['user_id']
            session['username'] = user['username']
            session['is_admin'] = user['is_admin']

            next_page = request.args.get('next', url_for('index'))
            conn.close()
            return redirect(next_page)
        else:
            conn.close()
```

```
            return render_template("login.html", error="Invalid username or password")

    return render_template("login.html")

@app.route("/register", methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('password')
        email = request.form.get('email')

        if not all([username, password, email]):
            return render_template("register.html", error="All fields are required")

        conn = get_db_connection()

        # Check if username or email already exists
        existing_user = conn.execute('''
        SELECT * FROM users WHERE username = ? OR email = ?
        ''', (username, email)).fetchone()

        if existing_user:
            conn.close()
            return render_template("register.html",
                                   error="Username or email already exists")

        # Create new user
        conn.execute('''
        INSERT INTO users (username, password, email, is_admin)
```

```
        VALUES (?, ?, ?, 0)
        ''', (username, password, email))

        conn.commit()

        # Log the user in
        user = conn.execute('SELECT * FROM users WHERE username = ?',
(username,)).fetchone()
        session['user_id'] = user['user_id']
        session['username'] = user['username']
        session['is_admin'] = user['is_admin']

        conn.close()

        return redirect(url_for('index'))

    return render_template("register.html")

@app.route("/logout")
def logout():
    session.clear()
    return redirect(url_for('index'))

@app.route("/admin")
def admin_dashboard():
    if 'user_id' not in session or not session.get('is_admin'):
        return render_template("error.html", message="Unauthorized access"), 403

    conn = get_db_connection()
```

```
    # Get summary data
    book_count = conn.execute('SELECT COUNT(*) as count FROM books').fetchone()
['count']
    user_count = conn.execute('SELECT COUNT(*) as count FROM users').fetchone()['count']
    order_count = conn.execute('SELECT COUNT(*) as count FROM orders').fetchone()
['count']
    total_sales = conn.execute('SELECT SUM(total_amount) as total FROM
orders').fetchone()['total'] or 0


    # Get recent orders
    recent_orders = conn.execute('''
    SELECT o.order_id, o.order_date, o.status, o.total_amount, u.username
    FROM orders o
    JOIN users u ON o.user_id = u.user_id
    ORDER BY o.order_date DESC LIMIT 5
    ''').fetchall()


    # Get low stock books
    low_stock_books = conn.execute('''
    SELECT b.book_id, b.title, b.stock
    FROM books b
    WHERE b.stock < 5
    ORDER BY b.stock ASC
    ''').fetchall()


    conn.close()


    return render_template("admin_dashboard.html",
                           book_count=book_count,
                           user_count=user_count,
```

```
                order_count=order_count,
                total_sales=total_sales,
                recent_orders=recent_orders,
                low_stock_books=low_stock_books)

@app.route("/admin/books")
def admin_books():
    if 'user_id' not in session or not session.get('is_admin'):
        return render_template("error.html", message="Unauthorized access"), 403

    conn = get_db_connection()
    books = conn.execute('''
    SELECT b.book_id, b.title, b.stock, b.price, a.name as author_name
    FROM books b
    JOIN authors a ON b.author_id = a.author_id
    ORDER BY b.book_id
    ''').fetchall()

    conn.close()

    return render_template("admin_books.html", books=books)

@app.route("/admin/edit_book/<int:book_id>", methods=['GET', 'POST'])
def admin_edit_book(book_id):
    if 'user_id' not in session or not session.get('is_admin'):
        return render_template("error.html", message="Unauthorized access"), 403

    conn = get_db_connection()
```

```
    if request.method == 'POST':
        title = request.form.get('title')
        author_id = request.form.get('author_id')
        stock = request.form.get('stock')
        price = request.form.get('price')
        published_date = request.form.get('published_date')
        information = request.form.get('information', '')

        if not all([title, author_id, stock, price, published_date]):
            authors = conn.execute('SELECT * FROM authors').fetchall()
            book = conn.execute('SELECT * FROM books WHERE book_id = ?',
(book_id,)).fetchone()
            conn.close()
            return render_template("admin_edit_book.html", book=book, authors=authors,
                                   error="All fields except Information are required")

        # Update book
        conn.execute('''
        UPDATE books
        SET title = ?, author_id = ?, stock = ?, price = ?, published_date = ?, information = ?
        WHERE book_id = ?
        ''', (title, author_id, stock, price, published_date, information, book_id))

        # Update categories
        conn.execute('DELETE FROM books_categories WHERE book_id = ?', (book_id,))
        categories = request.form.getlist('categories')
        for category_id in categories:
            conn.execute('INSERT INTO books_categories (book_id, category_id) VALUES (?, ?)',
                         (book_id, category_id))
```

```
        conn.commit()
        conn.close()

        return redirect(url_for('admin_books'))

    book = conn.execute('SELECT * FROM books WHERE book_id = ?',
(book_id,)).fetchone()
    if not book:
        conn.close()
        return render_template("error.html", message="Book not found"), 404

    authors = conn.execute('SELECT * FROM authors').fetchall()
    categories = conn.execute('SELECT * FROM categories').fetchall()

    # Get selected categories
    book_categories = conn.execute('''
    SELECT category_id FROM books_categories WHERE book_id = ?
    ''', (book_id,)).fetchall()
    selected_categories = [cat['category_id'] for cat in book_categories]

    conn.close()

    return render_template("admin_edit_book.html", book=book, authors=authors,
                           categories=categories, selected_categories=selected_categories)

@app.route("/admin/orders")
def admin_orders():
    if 'user_id' not in session or not session.get('is_admin'):
```

```
        return render_template("error.html", message="Unauthorized access"), 403

    status_filter = request.args.get('status', '')

    conn = get_db_connection()

    query = '''
    SELECT o.*, u.username
    FROM orders o
    JOIN users u ON o.user_id = u.user_id
    '''
    params = []

    if status_filter:
        query += ' WHERE o.status = ?'
        params.append(status_filter)

    query += ' ORDER BY o.order_date DESC'

    orders = conn.execute(query, params).fetchall()
    conn.close()

    return render_template("admin_orders.html", orders=orders, selected_status=status_filter)

@app.route("/admin/update_order/<int:order_id>", methods=['POST'])
def admin_update_order(order_id):
    if 'user_id' not in session or not session.get('is_admin'):
        return render_template("error.html", message="Unauthorized access"), 403
```

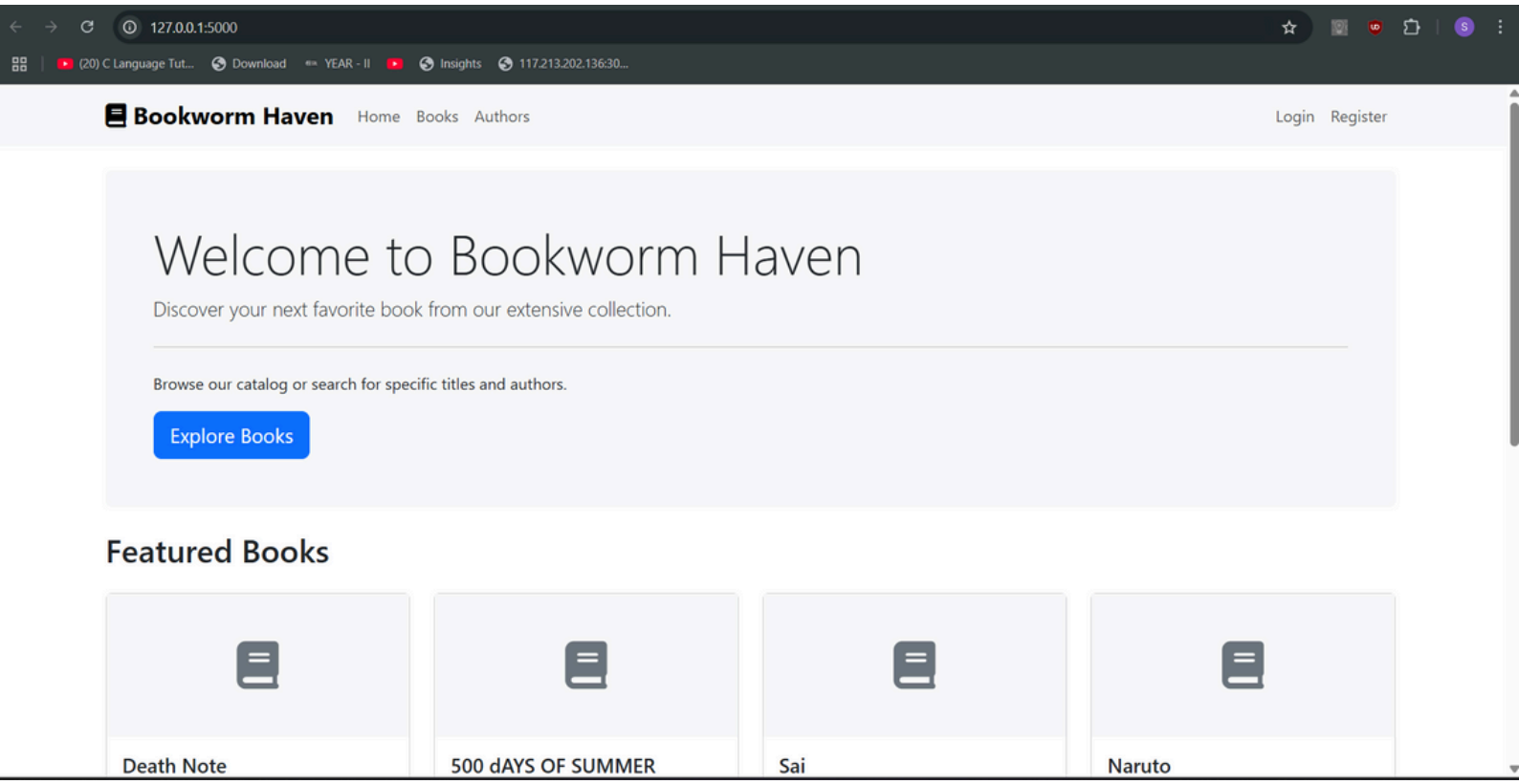
```
    status = request.form.get('status')

    conn = get_db_connection()
    conn.execute('UPDATE orders SET status = ? WHERE order_id = ?', (status, order_id))
    conn.commit()
    conn.close()

    return redirect(url_for('admin_orders'))

@app.route("/api/books")
def api_books():
    conn = get_db_connection()
    books = conn.execute("'
    SELECT b.book_id, b.title, b.price, a.name as author_name
    FROM books b
    JOIN authors a ON b.author_id = a.author_id
    ORDER BY b.book_id
    "').fetchall()

    # Convert to list of dicts
    books_list = [dict(b) for b in books]

    conn.close()

    return jsonify(books_list)

@app.route("/admin/sql_data")
def sql_data():
    if 'user_id' not in session or not session.get('is_admin'):
```

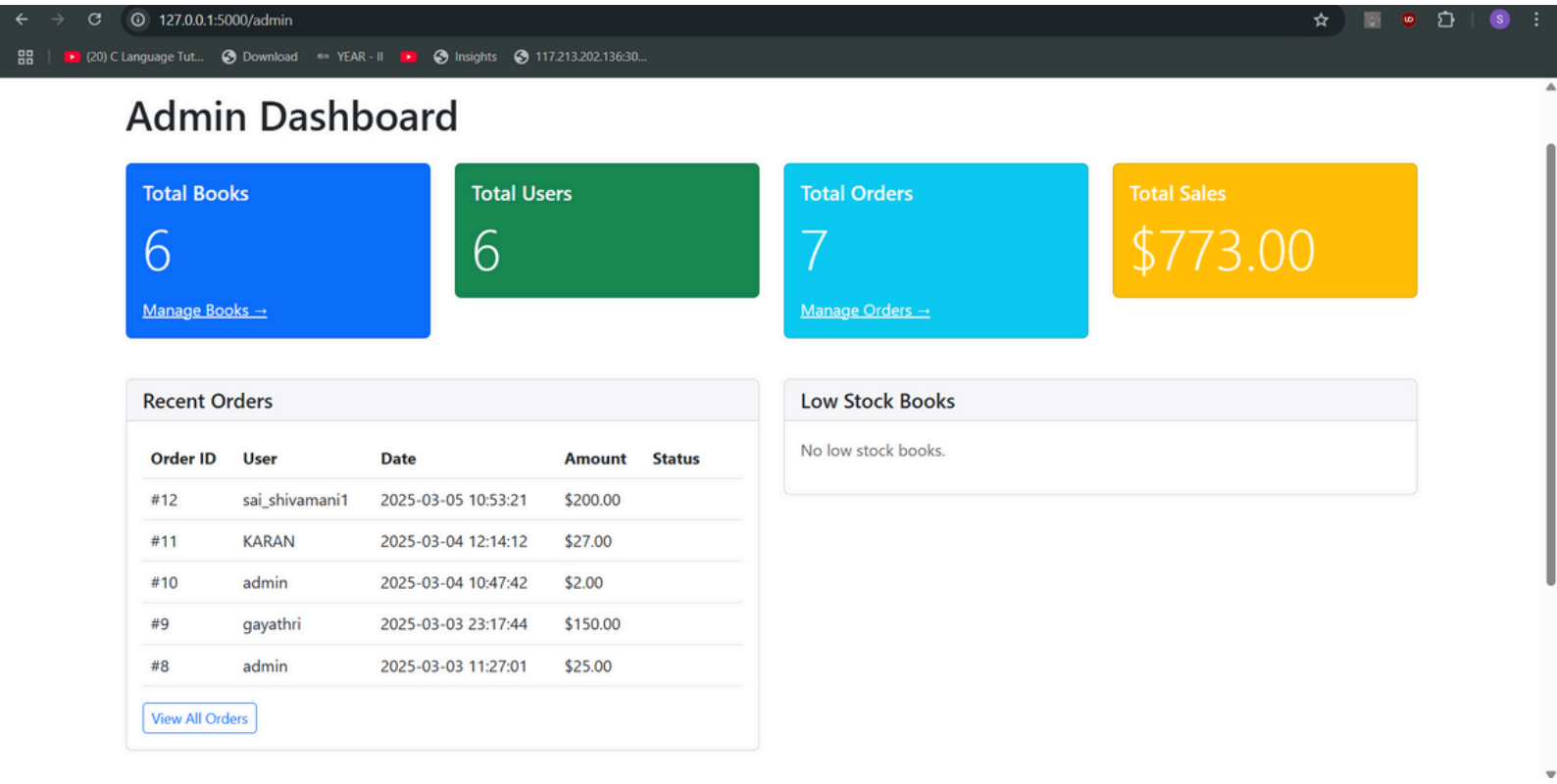
```
        return render_template("error.html", message="Unauthorized access"), 403

    conn = get_db_connection()

    # Get all table names
    tables = conn.execute("SELECT name FROM sqlite_master WHERE type='table'").fetchall()

    # Get data from each table
    table_data = {}
    for table in tables:
        table_name = table['name']
        # Get column names and primary key info
        columns = conn.execute(f"PRAGMA table_info({table_name})").fetchall()
        column_names = [col['name'] for col in columns]

        # Identify primary key columns
        primary_keys = [col['name'] for col in columns if col['pk'] == 1]

        # Get all rows
        rows = conn.execute(f"SELECT * FROM {table_name}").fetchall()

        table_data[table_name] = {
            'columns': column_names,
            'rows': [dict(row) for row in rows],
            'primary_keys': primary_keys
        }

    conn.close()
```

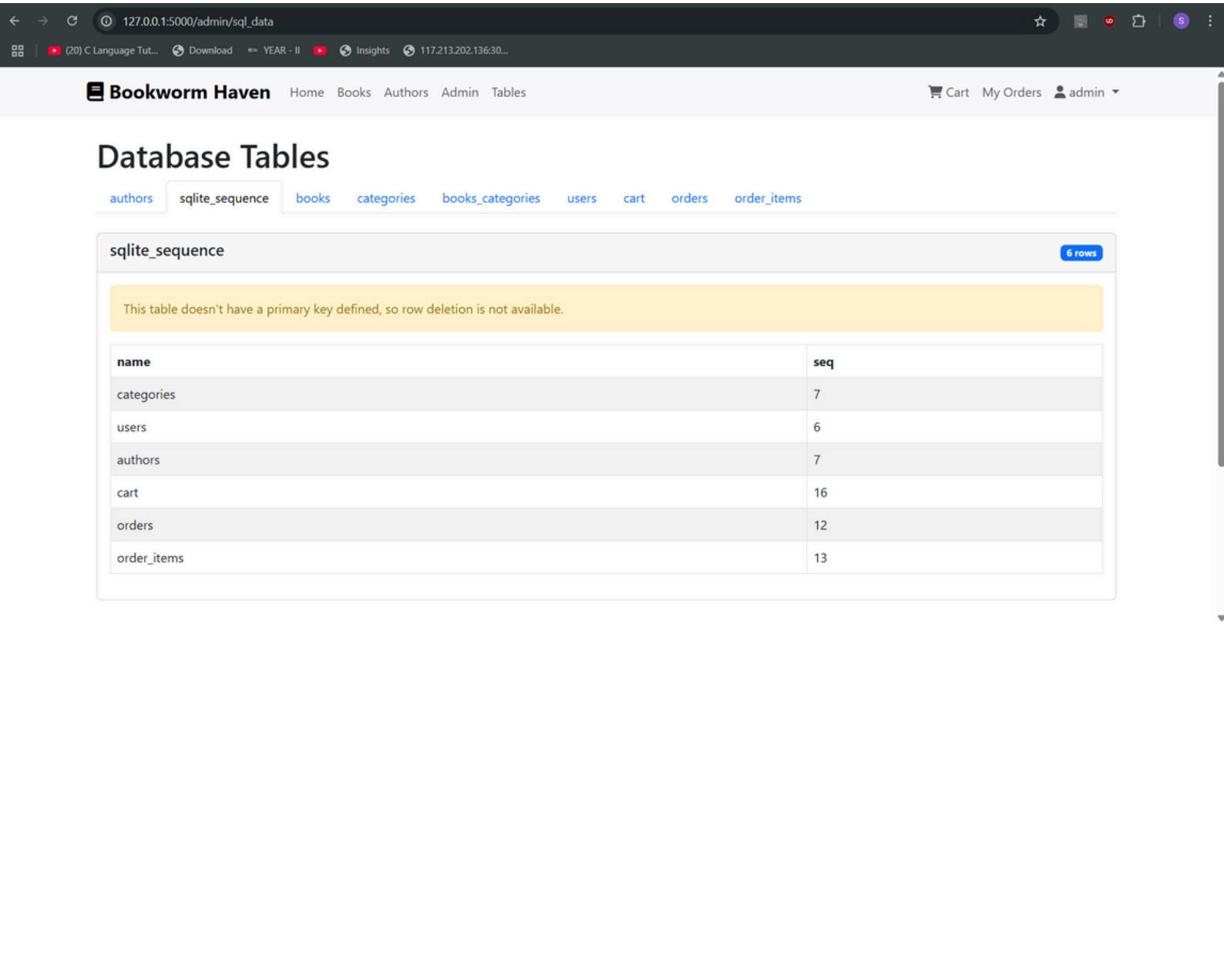
```
    return render_template("sql_data.html", tables=tables, table_data=table_data)

@app.route("/admin/delete_rows/<table_name>", methods=['POST'])
def delete_rows(table_name):
    if 'user_id' not in session or not session.get('is_admin'):
        return render_template("error.html", message="Unauthorized access"), 403

    # Get the primary key from the form
    primary_key = request.form.get('primary_key')

    # Get all selected rows (their primary key values)
    selected_rows = request.form.getlist('selected_rows')

    if not primary_key or not selected_rows:
        return render_template("error.html", message="No rows selected or missing primary key information"), 400

    conn = get_db_connection()

    try:
        # Use parameterized query with multiple values
        placeholders = ','.join(['?'] * len(selected_rows))
        query = f"DELETE FROM {table_name} WHERE {primary_key} IN ({placeholders})"

        conn.execute(query, selected_rows)
        conn.commit()
        conn.close()
        return redirect(url_for('sql_data'))
```

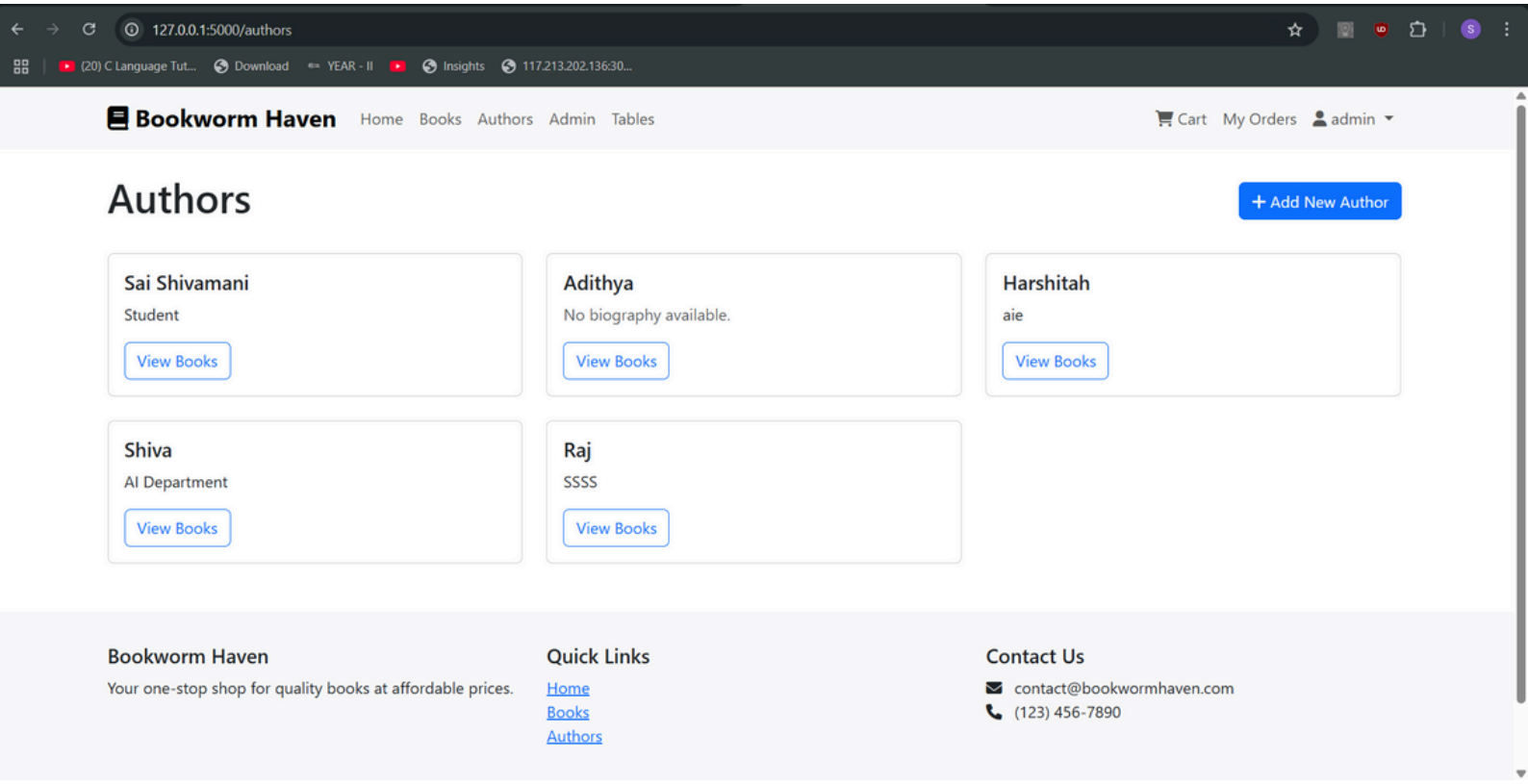
```
    except sqlite3.Error as e:
        conn.close()
        return render_template("error.html", message=f"Database error: {str(e)}"), 500

if __name__ == "__main__":
    app.run(debug=True)
```

4.Visual Representation:

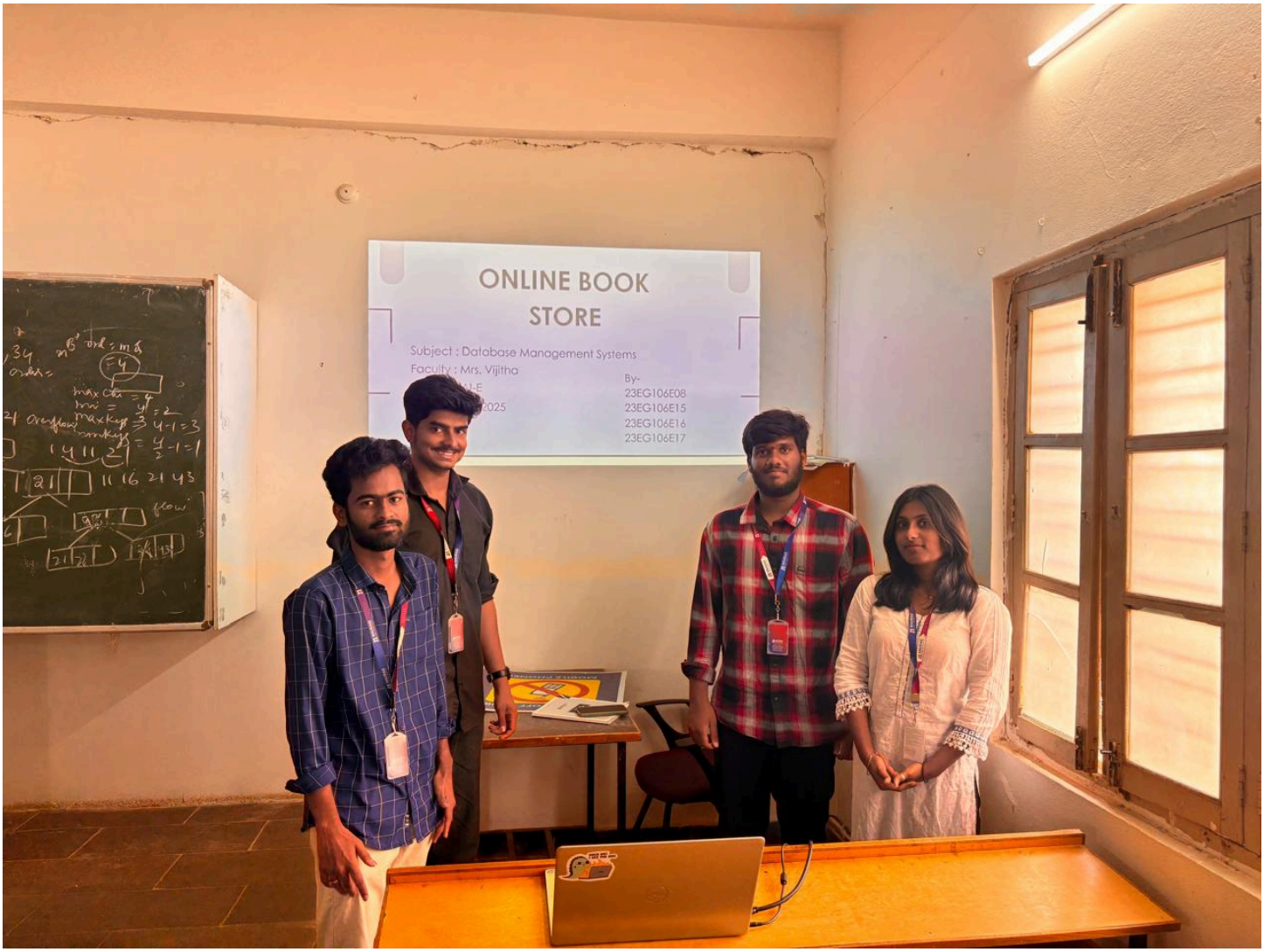



127.0.0.1:5000/admin
(20) C Language Tut...
Download
YEAR - II
Insights
117.213.202.136:30...
Admin Dashboard
Total Books
6
Manage Books →
Total Users
6
Total Orders
7
Manage Orders →
Total Sales
$773.00
Recent Orders
Order ID User Date Amount Status
#12 sai_shivamani1 2025-03-05 10:53:21 $200.00
#11 KARAN 2025-03-04 12:14:12 $27.00
#10 admin 2025-03-04 10:47:42 $2.00
#9 gayathri 2025-03-03 23:17:44 $150.00
#8 admin 2025-03-03 11:27:01 $25.00
View All Orders
Low Stock Books
No low stock books.

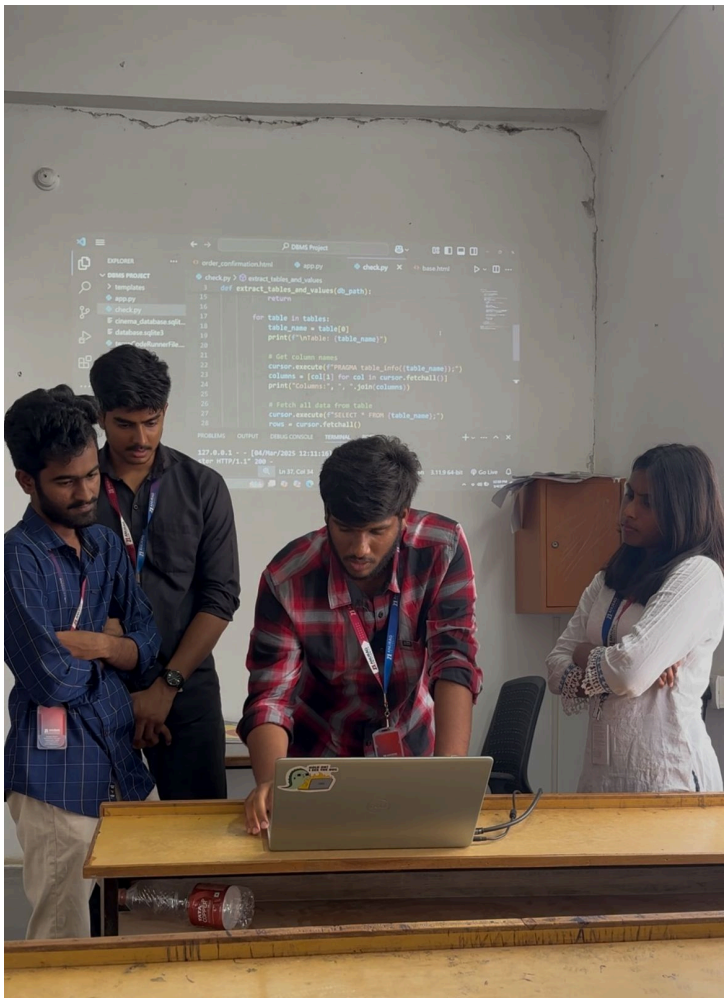

127.0.0.1:5000/admin/sql_data
(20) C Language Tut...
Download
YEAR - II
Insights
117.213.202.136:30...
Bookworm Haven
Home Books Authors Admin Tables
Cart My Orders admin
Database Tables
authors sqlite_sequence books categories books_categories users cart orders order_items
sqlite_sequence
6 rows
This table doesn't have a primary key defined, so row deletion is not available.
name seq
categories 7
users 6
authors 7
cart 16
orders 12
order_items 13

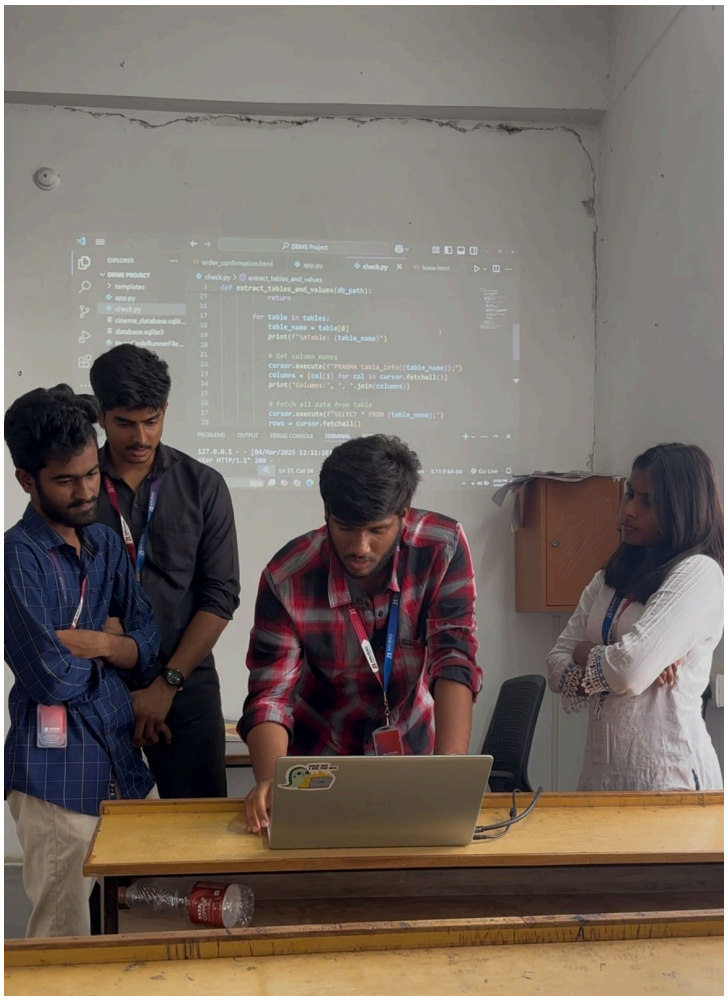

127.0.0.1:5000/authors
(20) C Language Tut...
Download
YEAR - II
Insights
117.213.202.136:30...
Bookworm Haven
Home
Books
Authors
Admin
Tables
Cart
My Orders
admin
Authors
+ Add New Author
Sai Shivamani
Student
View Books
Adithya
No biography available.
View Books
Harshitah
aie
View Books
Shiva
AI Department
View Books
Raj
SSSS
View Books
Bookworm Haven
Your one-stop shop for quality books at affordable prices.
Quick Links
Home
Books
Authors
Contact Us
contact@bookwormhaven.com
(123) 456-7890

Glimpses:


ONLINE BOOK
STORE
Subject : Database Management Systems
Faculty : Mrs. Vijitha
By-
23EG106E08
23EG106E15
23EG106E16
23EG106E17


ONLINE BOOK
STORE
Subject : Database Management Systems
Faculty : Mrs. Vijitha
Section : AI-E
By-
23EG106E08
23EG106E15
23EG106E16
23EG106E17


DBMS Project
EXPLORER
DBMS PROJECT
templates
app.py
check.py
cinema_database.sqlit...
database.sqlite3
order_confirmation.html
app.py
check.py
base.html
def extract_tables_and_values(db_path):
return
for table in tables:
table_name = table[0]
print(f"\nTable: {table_name}")
# Get column names
cursor.execute(f"PRAGMA table_info({table_name});")
columns = [col[1] for col in cursor.fetchall()]
print("Columns:", ", ".join(columns))
# Fetch all data from table
cursor.execute(f"SELECT * FROM {table_name};")
rows = cursor.fetchall()
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
Go Live


DBMS Project
EXPLORER
DBMS PROJECT
templates
app.py
check.py
cinema_database.sqlit...
database.sqlite3
order_confirmation.html
app.py
check.py
base.html
def extract_tables_and_values(db_path):
return
for table in tables:
table_name = table[0]
print(f"\nTable: {table_name}")
# Get column names
cursor.execute(f"PRAGMA table_info({table_name});")
columns = [col[1] for col in cursor.fetchall()]
print("Columns:", ", ".join(columns))
# Fetch all data from table
cursor.execute(f"SELECT * FROM {table_name};")
rows = cursor.fetchall()
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
Go Live

Conclusion on the Database Design:

The **Bookstore Management System** database is well-structured and efficiently handles user authentication, book inventory, and order processing. Below are the key takeaways:

1. Strengths of the Database Design:

✅ **Normalized Structure**: The database follows **normalization principles**, reducing redundancy and ensuring data integrity.
✅ **Efficient Many-to-Many Relationships**: The books_categories table efficiently manages book categorization.
✅ **Scalability**: The structure supports adding new features like **wishlists, reviews, and payment methods**.
✅ **Session-Based User Management**: Secure user authentication with role-based access for admins.

2. Future Enhancements

🚀 **REST API Integration**: Expose APIs for mobile apps or third-party integrations.
🚀 **Payment Gateway**: Add Stripe or PayPal for seamless online payments.
🚀 **Analytics Dashboard**: Track best-selling books, user activity, and revenue trends.

Final Verdict:

The current database **effectively supports the bookstore's core operations**, but with **minor improvements** (security enhancements, indexing, and analytics), it can become a **fully scalable and secure e-commerce platform**.