Project Step 5

Project Title: Beaverton Community Library

Team Members:

- Anil Verman
- Alexander Lott

Website URL: http://classwork.engr.oregonstate.edu:51320/

Executive Summary:

The Beaverton Community Library database project began as a simple plan to track members, books, and checkouts. Over time, and with consistent feedback, it has developed into a fully working React-based website with a clean, easy-to-use interface and complete functionality.

At the start, our focus was on building the database structure and connecting all tables. Early feedback encouraged us to add more details about the library, so we included figures for patrons, monthly checkouts, total books, genres, and authors. We also updated table relationships to make them more realistic, such as changing Books–Genres to a many-to-many relationship. Naming was standardized to camelCase with 'ID' at the end, and we added fields like 'year' and 'isbn' to the Books table.

During the schema and SQL phase, we ensured capitalization was consistent across the ERD, outline, and database. We documented missing intersection tables and added cascading update/delete rules where needed. Intersection tables were confirmed to be correctly formed and normalized.

Earlier versions of the site displayed headers without working data or full CRUD functions. Based on feedback, we implemented JOIN-based SELECT queries with clear column names, added INSERT forms, made DELETE and UPDATE functions work for all relevant tables, and included sample data for testing. All main tables and intersection tables now display and update correctly.

We used AI tools in a limited way for brainstorming SQL queries, and debugging issues with our production React server by pasting in error messages and following suggested fixes. While AI sped up some tasks, we often needed to adapt the output to fit our exact needs. The most valuable improvements came from classmates testing the live site and providing direct, specific feedback.

Through these steps, our project has evolved into a complete, stable, and user-friendly system that meets all requirements and is ready for final delivery.

Overview:

Beaverton Community Library is a relatively small, yet beloved public library situated within the quaint town of Beaverton, OR. Due to an unexpected clerical error, funding earmarked for the expansion of a much more prosperous library in the state capital was serendipitously delivered to Beaverton Community Library instead, allowing it to finally expand its book catalogue and renovate its computer systems as the populace of Beaverton demanded. However, these much-anticipated updates to the library's facilities have caused a surge of new patrons to materialize, to the point that the library's pen-and-paper member tracking system can no longer track them all. In order to avoid losing track of its members' activities and its books, and in order to ensure that Beaverton Community Library's tracking system could be scaled up to meet the needs of its ever-expanding clientele, the local government ordered the development of a database through which the library's operations may be easily managed and recorded.

The database setup outlined below is intended to be a relatively simple, yet robust one capable of tracking the library's hundreds of unique patrons (roughly 350 patrons in total who collectively check out about 2,200 books per month) and the almost 8,000 books that the library has available for them to read. Each member can have multiple books checked out at the same time, but each book can only be checked out by one member at a time. Additionally, since the library's books are organized by genre (the library carries ~20 different genres), each book must be assigned at least one genre by the database; each book also has a corresponding author (the library carries works from about 6,000 different authors) who is tracked by the database in a separate table. The database also has dedicated tables for tracking both the checkout status (i.e. which members checked out which books, the books' due dates, their checkout dates, etc.) of the library's books and the reviews (i.e. ratings on a scale of 1 to 5) that the library's members have submitted for them, allowing both librarians and patrons to get a sense of which books are garnering the most traffic.

Project and Database Outline:

- **Members:** records the details of Members of this library
    - memberID: int, auto_increment, not NULL, PK
    - name: varchar, not NULL
    - email: varchar, not NULL
    - phone: varchar, not NULL
    - Relationships:
        - M:M relationship between Members (optional) and Books (optional) is implemented with the intersection table Reviews
        - M:M relationship between Members (optional) and Books (optional) is implemented with the intersection table Checkouts

- **Books:** records the details of Books at this library
    - bookID: int, auto_increment, not NULL, PK
    - title: varchar, not NULL
    - authorID: int, not NULL
    - year: int, not NULL
    - isbn: varchar, not NULL
    - Relationships:
        - M:M relationship between Books (optional) and Members (optional) is implemented with the intersection table Reviews
        - M:M relationship between Books (optional) and Members (optional) is implemented with the intersection table Checkouts
        - M:M relationship between Books (mandatory) and Genres (optional) is implemented with the intersection table Books_has_Genres (a book can have multiple genres)
        - M:1 relationship between Books (mandatory) and Authors (mandatory, multiple books can be of the same author)

- **Genres:** lists all Genres of books
    - genreID: int, auto_increment, not NULL, PK
    - description: varchar, not NULL
    - Relationships:
        - M:M relationship between Books (mandatory) and Genres (optional) is implemented with the intersection table Books_has_Genres (a book can have multiple genres)

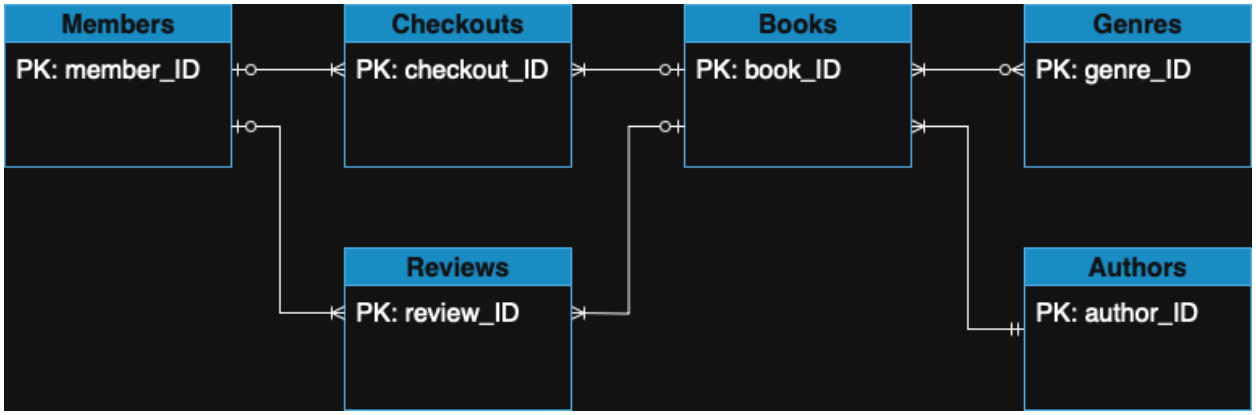
- **Books_has_Genres:** lists which books are associated with which genres, and vice versa
    - bookID: int, auto_increment, not NULL
    - genreID: int, auto_increment, not NULL
    - Relationships:
        - Mediates an M:M relationship between Books (mandatory) and Genres (optional)

- **Authors:** lists all Authors of books
    - authorID: int, auto_increment, not NULL, PK
    - name: varchar, not NULL
    - Relationships:
        - 1:M relationship between Authors (mandatory) and Books (mandatory, multiple books can be of the same author)

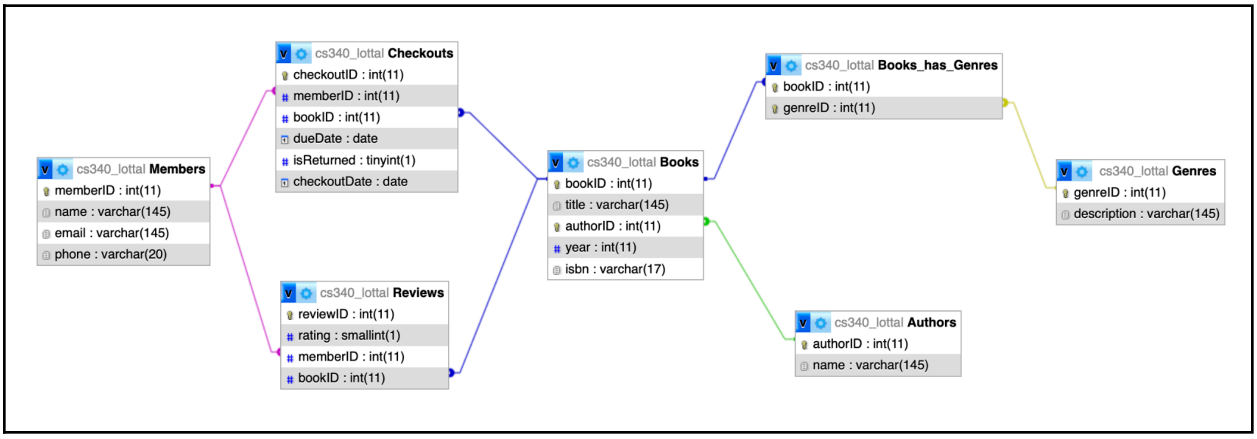
- **Checkouts:** records which member checked out which book. Also records the check-out date, due date, and whether or not the book has been returned.
    - checkoutID: int, auto_increment, not NULL, PK
    - memberID: int, not NULL
    - bookID: int, not NULL
    - dueDate: date, not NULL
    - isReturned: tinyint, not NULL
    - checkoutDate: date, not NULL
    - Relationships:
        - This is an intersection table between Books and Members
        - M:1 relationship between Checkouts (mandatory) and Members (optional)
        - M:1 relationship between Checkouts (mandatory) and Books (optional)

- **Reviews:** records which members rated which book. The rating is a number between 1 and 5.
    - reviewID: int, auto_increment, not NULL, PK
    - rating: smallint, not NULL
    - memberID: int, not NULL
    - bookID: int, not NULL
    - Relationships:
        - This is an intersection table between Books and Members
        - M:1 relationship between Reviews (mandatory) and Members (optional)
        - M:1 relationship between Reviews (mandatory) and Books (optional)

Entity-Relationship Diagram:



Schema Diagram:



Sample Data:

```
INSERT INTO `authors` (`authorID`, `name`) VALUES
(1, 'George Orwell'),
(2, 'Andrew Hunt'),
(3, 'J.R.R. Tolkien');
```

```
INSERT INTO `books` (`bookID`, `title`, `authorID`, `year`, `isbn`) VALUES
(1, '1984', 1, 1949, '9780451524935'),
(2, 'The Pragmatic Programmer', 2, 1999, '9780201616224'),
(3, 'The Hobbit', 3, 1937, '9780547928227');
```

```
INSERT INTO `genres` (`genreID`, `description`) VALUES
(1, 'Dystopian'),
(2, 'Software Development'),
(3, 'Fantasy');
```

```
INSERT INTO `books_has_genres` (`bookID`, `genreID`) VALUES
(1, 1),
(2, 2),
(3, 3);
```

```
INSERT INTO `members` (`memberID`, `name`, `email`, `phone`) VALUES
(1, 'Alice Smith', 'alice@example.com', '123-456-7890'),
(2, 'Bob Johnson', 'bob@example.com', '987-654-3210'),
(3, 'Carol Davis', 'carol@example.com', '555-123-4567');
```

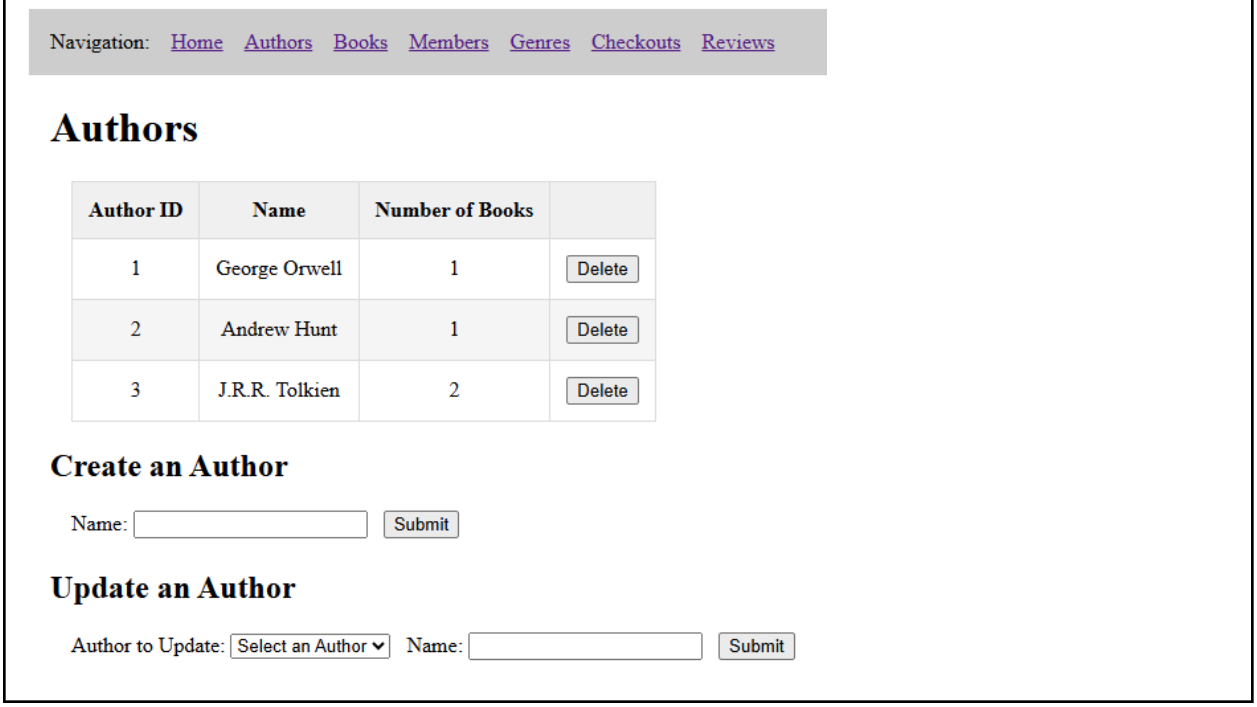
```
INSERT INTO `checkouts` (`checkoutID`, `memberID`, `bookID`, `dueDate`,
`isReturned`, `checkoutDate`) VALUES
(1, 1, 1, '2025-08-01', 0, '2025-07-15'),
(2, 2, 2, '2025-08-05', 1, '2025-07-10'),
(3, 3, 3, '2025-08-10', 0, '2025-07-20');
```

```
INSERT INTO `reviews` (`reviewID`, `rating`, `memberID`, `bookID`) VALUES
(1, 5, 1, 1),
(2, 4, 2, 2),
(3, 5, 3, 3);
```

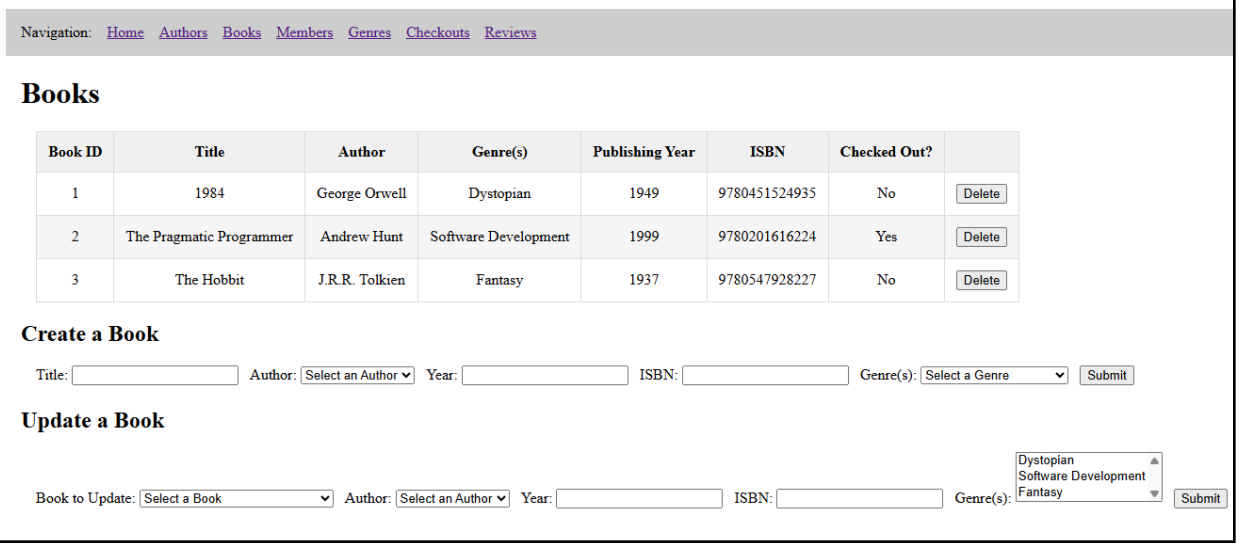
UI Screenshots:

RESET DB Home page
Clicking the Reset Database button will reset the database back to its sample data

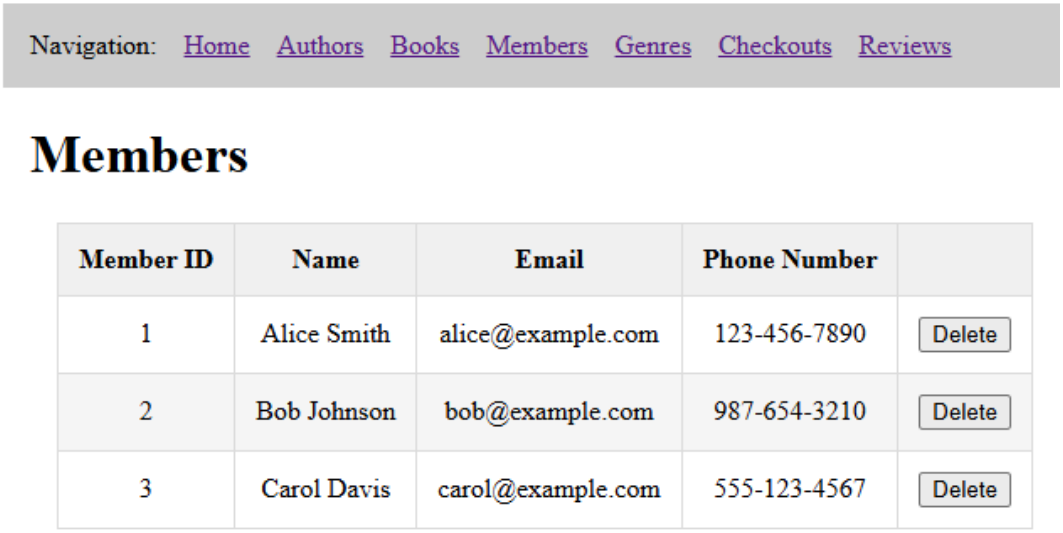
READ, INSERT, UPDATE, DELETE, DYNAMIC DROPDOWN Authors page



READ, INSERT, UPDATE, DELETE, DYNAMIC DROPDOWN Books page
INSERT, UPDATE, DELETE M:N Books_has_Genres table (via Genre(s) Update field)
Genre(s) Update field allows for multi-select with Cmd+Click



READ, DELETE Members page



READ, DELETE Genres page

Navigation: Home Authors Books Members Genres Checkouts Reviews

Genres

Genre ID	Description	Number of Books	
1	Dystopian	1	Delete
2	Software Development	2	Delete
3	Fantasy	1	Delete

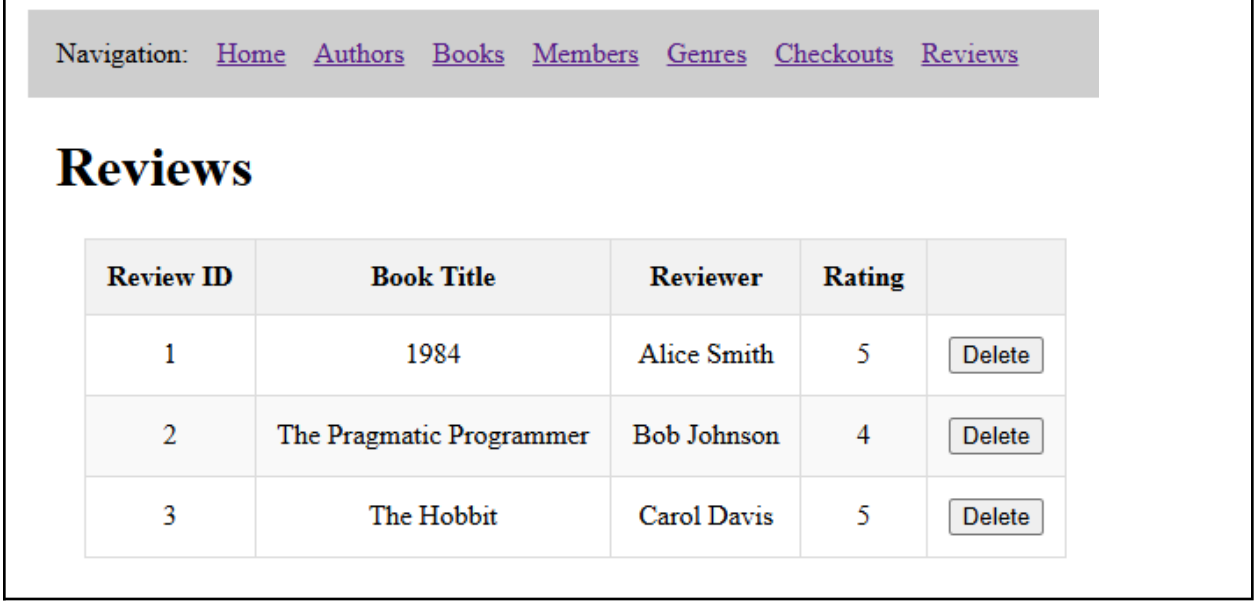
READ, DELETE Checkouts page;
UPDATE M:N Checkouts table (via CASCADE from Members and Books tables)

Navigation: Home Authors Books Members Genres Checkouts Reviews

Checkouts

Checkout ID	Checked Out By	Book Title	Checkout Date	Due Date	Returned?	
1	Alice Smith	1984	2025-07-15T07:00:00.000Z	2025-08-01T07:00:00.000Z	No	Delete
2	Bob Johnson	The Pragmatic Programmer	2025-07-10T07:00:00.000Z	2025-08-05T07:00:00.000Z	Yes	Delete
3	Carol Davis	The Hobbit	2025-07-20T07:00:00.000Z	2025-08-10T07:00:00.000Z	No	Delete

READ, DELETE Reviews page;
UPDATE M:N Reviews table (via CASCADE from Members and Books tables)



Citations:

- Used draw.io to create the Entity-Relationship diagram
- Used phpMyAdmin to create the schema diagram
- Used ChatGPT to edit this document.