BookPilot

Meiru Han, Pu Tan, Tianyi Wei, Yuan Zhuang

Spring 2023

Introduction

Our project aims to create a web application that streamlines and simplifies the process of discovering and exploring books, authors, and reviews, making it easier for users to find their next great read. Our target problem is the difficulty of navigating the vast literary landscape and finding books and authors that align with individual preferences. We have designed our application to provide a comprehensive yet intuitive platform that enables users to search and explore books, authors, and reviews efficiently and effectively.

The motivation behind our project stems from our appreciation for literature and the desire to make the discovery and exploration of books and authors a more enjoyable and accessible experience for users. Inspired by the Spotify-themed web application from homework 2, we envisioned a platform that not only bridges data on books, authors, and reviews but also fosters a sense of curiosity and excitement as users delve into the world of literature.

Architecture

Technologies

React is used for the front-end of the website, which includes the presentation layer, user interaction, and data display. The back-end of the website is powered by a MySQL database, which is hosted on Amazon Web Services (AWS). The database stores all the information needed for the website, including author data, book data, and review data. The communication between the front-end and back-end is handled by Node.js.

Applications

There are three pages on the website: Home Page, Books Page, and Authors Page.

On top of the home page, there are two sign in options: sign in with Google or sign in with Twitter. Under the sign-in options, there are "Book of the Day" and "Author of the Day" features that pick a random book and author for the users so that users who are unsure of what they are looking for can check out the book or the books the author wrote to get started. We also built a search bar for users to search books or authors by title or name. The search result is linked to the Books page or the Authors page depending on the user input. The home page lists top 100 popular books in a table. There are two ways to define popular books: the users can choose to order by number of ratings or the average rating scores of each book. The home page also hosts 5 rankings that show the top 10 popular books in each of the 5 most popular genres. The last table on the home page displays the most representative book for each author.

The Books Page has some search bars and sliders on the top for users to search by title, author, genre, publisher and filter by published date, rating scores, and number of ratings. If no search criterion is given, the page displays all books in a table in alphabetic order by the book titles. Each book has its own page with book information (author, genre, etc), a cover picture, descriptions (if provided in the dataset), rating scores, and reviews in texts.

There are also search bars and sliders on the top of the Authors Page. Users can search authors by author's name, country, and genre. Users can further filter the results by specifying the fan counts and rating scores using the sliders. Each author has their own page with their profile picture, a list of books they wrote, and some relevant statistics regarding popularity.

Data

We used three datasets to build the website:

1. Books dataset

   This dataset contatins 212404 unique books on Google Books published or plan to publish before 2030. The attributes we used from this dataset included: Title, description, authors, image, previewLink, publisher, publishedDate, infoLink, and categories (genres). Some major data cleaning and normalization on this dataset included:

   (a) Remove all books that either don't have a review in the review dataset or don't have an author in the author dataset.

   (b) Extract all genres of all books (normalized to all lowercase letters).

   (c) Get individual authors for each book by exploding the "authors" column

   Figure 1 shows some info for this dataset, and Figure 2 shows the top 10 genres with the number of books in each genre. Notice that one book can belong to multiple genres.

   Link: `https://www.kaggle.com/datasets/mohamedbakhet/amazon-books-reviews?select=books_data.csv`

```
RangeIndex: 212404 entries, 0 to 212403
Data columns (total 10 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   Title          212403 non-null  object
 1   description    143962 non-null  object
 2   authors        180991 non-null  object
 3   image          160329 non-null  object
 4   previewLink    188568 non-null  object
 5   publisher      136518 non-null  object
 6   publishedDate  187099 non-null  object
 7   infoLink       188568 non-null  object
 8   categories     171205 non-null  object
 9   ratingsCount   49752 non-null   float64
dtypes: float64(1), object(9)
memory usage: 16.2+ MB
```

Figure 1: Books Info

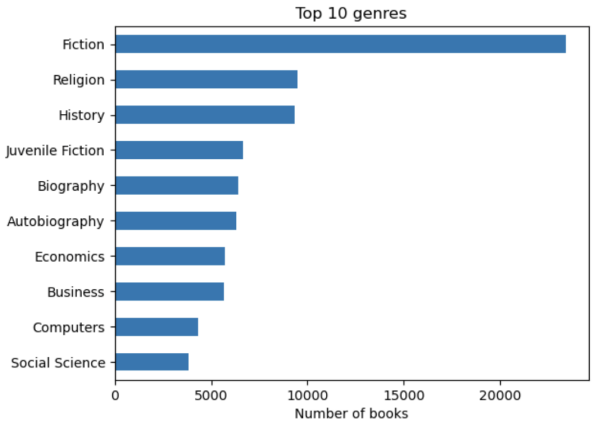

Figure 2: Top 10 genres

2. Reviews dataset

   This dataset contains 3M feedback (including ratings and reviews) from users on the 212404 unique books spaning May 1996 - July 2014. The attributes we used from the dataset included: Id (ISBN-10), Title, User_id, profileName, review/helpfulness, review/score, review/time, and review/text. Some major data cleaning steps on this dataset included:

   (a) Exclude rows with missing User id.

   (b) Remove duplicate review entries based on User id and time of review.

   (c) Remove all reviews that don't belong to any book in the book dataset.

   We used review/score to compute average rating for each book, and we used review/helpfulness and review/text to display helpful reviews for each book. Figure 3 shows some info for this dataset, and Figure 4 shows the distribution of book ratings. We can see that users who review books tend to give higher scores.

   Link: `https://www.kaggle.com/datasets/mohamedbakhet/amazon-books-reviews?select=Books_rating.csv`

```
RangeIndex: 3000000 entries, 0 to 2999999
Data columns (total 10 columns):
 #   Column              Dtype
---  ------              -----
 0   Id                  object
 1   Title               object
 2   Price               float64
 3   User_id             object
 4   profileName         object
 5   review/helpfulness  object
 6   review/score        float64
 7   review/time         int64
 8   review/summary      object
 9   review/text         object
dtypes: float64(2), int64(1), object(7)
memory usage: 228.9+ MB
```

Figure 3: Reviews Info

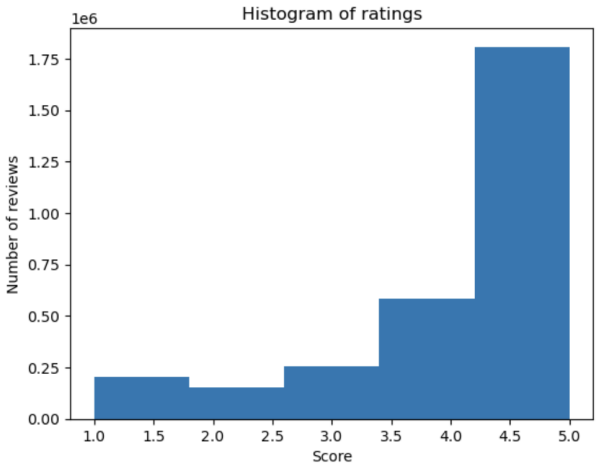

Figure 4: Distribution of book ratings

3. Authors dataset

This dataset contains 209500 authors on GoodRead. There are 20 attributes in the dataset, of which 7 of them are important to our website: authorid, name, fancount, image_url, ave_rating, country, and website. Some major data cleaning steps on this dataset included:

(a) Remove all authors that didn't write a book in the book dataset.

(b) Extract all genres in which authors write in (normalized to all lowercase letters).

We used this dataset along with the book dataset to design a webpage for each author displaying the books they wrote and relevant statistics. Figure 5 shows some info about this dataset, and Figure 6 shows the distribution of average author ratings.

Link: `https://www.kaggle.com/datasets/choobani/goodread-authors`

```
RangeIndex: 209517 entries, 0 to 209516
Data columns (total 20 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   authorid           209517 non-null  int64
 1   name               209517 non-null  object
 2   workcount          209517 non-null  int64
 3   fan_count          209517 non-null  int64
 4   gender             209517 non-null  object
 5   image_url          209517 non-null  object
 6   about              86724 non-null   object
 7   born               31230 non-null   object
 8   died               12488 non-null   object
 9   influence          7882 non-null    object
 10  average_rate       209517 non-null  float64
 11  rating_count       209517 non-null  int64
 12  review_count       209517 non-null  int64
 13  website            58320 non-null   object
 14  twitter            35122 non-null   object
 15  genre              73983 non-null   object
 16  original_hometown  44599 non-null   object
 17  country            44599 non-null   object
 18  latitude           44369 non-null   float64
 19  longitude          44369 non-null   float64
dtypes: float64(3), int64(5), object(12)
memory usage: 32.0+ MB
```

Figure 5: Authors Info

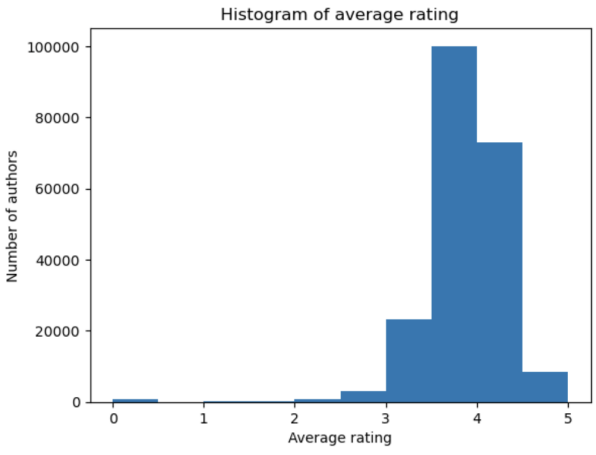

Figure 6: Distribution of average author ratings

Table	Cardinality
Book	78712
Author	35572
Genre	547
User	1008972
Review	1264776
Author_Of	83914
Belong_To	80906
Write_In	40868

Table 1: Cardinality of each table

Database

After cleaning the datasets, we split the 3 datasets into 8 tables (See Section ER Diagram and Relational Schema) for more usability. Table 1 shows the cardinality of each table in our database. More specifically, we created a table to store all genres with unique genre_id that we genreated and a table to store all user information with unique user_id that was already included in the Reviews dataset. We also appended a unique book_id to each book in the Books dataset. Additionally, because a book can be written by multiple authors and belong to multiple genres, we also created separate tables Author_Of, Write_In, and Belong_To to store relations between books, authors, and genres. Their many-to-many relations are reflected in the ER Diagram (See Figure 7).

ER Diagram and Relational Schema

1. Book(book_id,title,description,publisher,published_date,image,preview_link)
2. Author(author_id,name,fan_count,image_url, website,about,average_rate, rating_count, review_count)
3. User(user_id,name)
4. Review(user_id,book_id,time,text,score)

   FK: user_id foreign key referencing User(user_id)
   FK: book_id foreign key referencing Book(book_id)
5. Genre(genre_id,name)
6. Belong_to(book_id,genre_id)

   FK: book_id foreign key referencing Book(book_id)
   FK: genre_id foreign key referencing Genre(genre_id)
7. Write_in (author_id,genre_id)

   FK: author_id foreign key referencing Author(author_id)
   FK: genre_id foreign key referencing Genre(genre_id)
8. Author_of(author_id, book_id)

   FK: author_id foreign key referencing Author(author_id)
   FK: book_id foreign key referencing Book(book_id)

3NF Proof: The above schema is in 3NF. For all relations in the schema, all attributes are functionally dependent on the primary key(s) of their relation. There is no non-primary-key attribute that is transitively dependent on the primary key(s). Therefore, our schema is in 3NF.

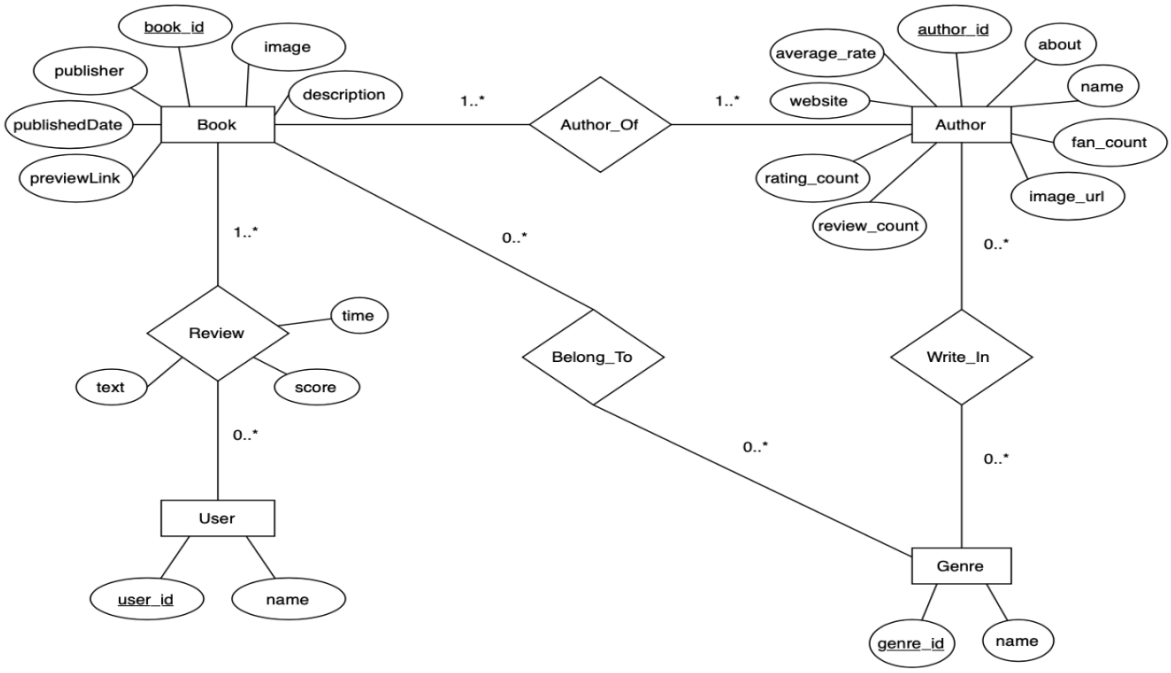

Figure 7: ER Diagram

Queries and Optimizations

Queries

1. **Top 10 books for each of the top 5 genres**: This query fetched from the database the top 10 books with the most reviews in each of the 5 most popular genres. Genre popularity were sorted by the number of books in each genre. This query was used to display the ranking tables for each of the 5 most popular genres on the Homepage.

```
WITH
  TOP5Genre AS (
  SELECT genre_id, COUNT(book_id) AS num_books
  FROM Belong_To
  GROUP BY genre_id
  ORDER BY COUNT(book_id) DESC
  LIMIT 5),
  top_books AS (
  SELECT
      Belong_To.genre_id, Book.book_id, Author.author_id, AVG(Review.score) AS average_score,
      COUNT(Review.user_id) AS num_rating,
      ROW_NUMBER() OVER (
      PARTITION BY Belong_To.genre_id ORDER BY COUNT(Review.user_id) DESC) AS rank_
  FROM Book
  INNER JOIN Author_Of ON Book.book_id = Author_Of.book_id
  INNER JOIN Author ON Author_Of.author_id = Author.author_id
  INNER JOIN Review ON Book.book_id = Review.book_id
  INNER JOIN Belong_To ON Book.book_id = Belong_To.book_id
  INNER JOIN TOP5Genre ON  Belong_To.genre_id = TOP5Genre.genre_id
  GROUP BY Belong_To.genre_id, Book.book_id, Author.author_id
```

```
  ORDER BY TOP5Genre.num_books DESC)
SELECT top_books.genre_id, Book.book_id, Book.title, Author.name AS author_name,
       Author.author_id, top_books.num_rating, top_books.average_score
FROM top_books
INNER JOIN Book ON top_books.book_id = Book.book_id
INNER JOIN Author ON top_books.author_id = Author.author_id
WHERE top_books.rank_ <= 10;
```

2. **Most representative book of popular authors**: This query fetched the most representative book of each author. It was used to create the table that shows the Most Representative Book of Each Author at the bottom of the Homepage.

```
WITH book_num_rating AS (
      SELECT b.book_id, b.title, COUNT(DISTINCT r.user_id) AS num_ratings
      FROM Book b JOIN Review r ON b.book_id = r.book_id
      GROUP BY b.book_id, b.title
  )
  SELECT a.author_id, a.name AS name, a.rating_count,a.average_rate, b1.book_id, b1.title AS title,
  FROM Author a JOIN Author_Of ao ON a.author_id = ao.author_id
      JOIN book_num_rating b1 ON ao.book_id = b1.book_id
      JOIN Belong_To bt on ao.book_id = bt.book_id
      JOIN Genre g on bt.genre_id = g.genre_id
  WHERE b1.num_ratings = (
      SELECT MAX(b2.num_ratings)
      FROM Author_Of ao2 JOIN book_num_rating b2 ON ao2.book_id = b2.book_id
      WHERE ao2.author_id = a.author_id
      )
  GROUP BY b1.book_id
  ORDER BY a.rating_count DESC, a.average_rate DESC
  LIMIT ${pageSize} OFFSET ${(page-1)*pageSize};
```

3. **Search books that match all specified conditions**: This query selected all books that match the requirements specified in the query parameters including title, author, genre, publisher, dateLow, dateHigh, ratingLow, ratingHigh, numratingLow, and numratingHigh. The search results were sorted in alphabetical order by the title of the books. The query enabled search by all these criterion on the Books page and also search by book title on the Homepage.

```
SELECT DISTINCT B.book_id, B.title, B.publisher, B.published_date, G.name AS genre,
       A.author_id, A.name AS author, R.score_avg, 2, R.num_ratings
FROM Book B
INNER JOIN Author_Of AO ON B.book_id = AO.book_id
INNER JOIN Author A ON AO.author_id = A.author_id
INNER JOIN Belong_To BT ON B.book_id = BT.book_id
INNER JOIN Genre G ON BT.genre_id = G.genre_id
INNER JOIN (
  SELECT book_id, ROUND(AVG(score),2) AS score_avg, COUNT(user_id) AS num_ratings
  FROM Review
  GROUP BY book_id
) R ON B.book_id = R.book_id
WHERE B.title LIKE '%${title}%'
AND A.name LIKE '%${author}%'
AND G.name LIKE '%${genre}%'
AND B.publisher LIKE '%${publisher}%'
```

```
AND B.published_date BETWEEN '${dateLow}' AND COALESCE('${dateHigh}', CURDATE())
AND R.score_avg >= ${ratingLow} AND R.score_avg <= ${ratingHigh}
AND R.num_ratings >= ${numratingLow} AND R.num_ratings <= ${numratingHigh}
ORDER BY B.title ASC;
```

4. **Calculate average rating of each book**: This query grouped reviews in the Review table by book_id and calculate the average rating of each book by averaging over all rating scores for that book. The resulting relation was sorted by the average rating in descending order and displayed on the Homepage. We picked the first 100 books to show on the Homepage.

```
WITH book_rating AS (
    SELECT Book.book_id, ROUND(AVG(Review.score),2) AS avg_rating
    FROM Review JOIN Book
    ON Review.book_id = Book.book_id
    GROUP BY Book.book_id)
SELECT Book.book_id, Book.title, book_rating.avg_rating
FROM Book JOIN book_rating
ON Book.book_id = book_rating.book_id
ORDER BY avg_rating DESC
LIMIT 100;
```

5. **Get info for each author**: This query fetched all information of an author requested by the author_id in the parameter. The information included every attribute in the Author data and all books the author has written.

```
SELECT A.author_id, A.name, A.fan_count, A.image_url, A.rating_count, A.website, A.about,
       A.average_rate, A.country, B.book_id, B.image, B.title
FROM Author A JOIN Author_Of AO
ON A.author_id = AO.author_id
JOIN Book B ON AO.book_id = B.book_id
JOIN Review R ON B.book_id = R.book_id
WHERE A.author_id = '${req.params.author_id}'
GROUP BY B.book_id
ORDER BY COUNT(R.user_id) DESC
```

Optimization

We mainly used techniques including adding indexes, removing unnecessary joins, and reducing the size of immediate relations to achieve query optimization. We added the following indexes in addition to the index on primary keys.

1. Book: title, published_date, publisher
2. Author: name, rating_count, average_rate
3. Genre: name
4. Review: score

These indexes improved the performance of queries that need to match specified conditions on one or more of these attributes and speed up the computation of the average scores. For reducing the size of immediate relations, we avoided joining with large tables (i.e. Review and User) at the begining of the query. Instead, we only joined with large tables after the necessary rows from other tables were selected.
We timed the execution time of the 4 complex queries before and after optimization (See Table 2).

Query Function	Runtime (pre-optimize)	Runtim (post-optimize)
Calculate average rating of each book	79s	3s
Search books that match all specified conditions	85s	3s
Top 10 books for each of the top 5 genres	95s	5s
Most representative book of popular authors	20s	12s

Table 2: Before and after optimization Execution time of complex queries

Technical Challenges

We indeed experienced many challenges during the course of the project since all team members are new to react and javascript. We listed a few technical challenges and how we addressed them below.

Data cleaning is very tedious and dirty work. We experienced many times that we thought our data cleaned using the pandas library was in the good shape, but DataGrip rejected our data for various reasons. For example, MySQL is case-insensitive while pandas is case-sensitive, so the unique keys in pandas might not be unique anymore when imported into MySQL database.

Query optimization is also a very challenging aspect of the project. When we wrote the queries, we already tried very hard to make the queries usable, and we found it very difficult to optimize them further with our knowledge. We ended up mainly adding indexes, removing unnecessary joins, and reducing the size of immediate relations to achieve query optimization.

Apart from the project itself, doing version control among 4 team members is another challenge. We are all not very familiar with git branching and thus we got a lot of conflicts on our branches at the begining. To address this challenge, we ended up doing frequent code review together to merge all our changes.

Extra Credit Features

For extra credit part, we add user login experience at the homepage including twitter sign in and google sign in two functions. Once you open the homepage, you can one of those two ways to sign in with your own account. After you sign in successfully, sign out button will replace the sign in button and you can choose to sign out.
In this part, we used firebase library to make it work. We created a firebase project and generated firebaseConfig from our project. After that, we initialized the firebase in an additional js file which is located in .../client/src/firebase.js. The next step is to create a Google API project and obtain OAuth 2.0 credentials so that you can use the "Client ID" and "Client Secret" provided by your Google API project to link with your firebase, which should be enabled in the Authentication part in your firebase project.
With all the preparation done, then we can import related library including signInWithPopup, signOut, onAuthStateChanged, googleProvider and so on from firebase to write our sign in and sign out functions. There are multiple ways to use sign in functions and we used signInWithPopup method from Firebase Auth, passing the auth instance and the googleProvider instance as arguments. This method displays a popup window for the user to sign in with their Google account. What's more, we added googleProvider.setCustomParameters part to Set custom parameters for the Google authentication provider. In this case, it sets the 'prompt' option to 'select account' to always show the account selection when signing in. Otherwise, you will always sign in with the same account even if you sign out or reopen the homepage.