Book For You

ZhiFan Xu, Chen Fan, Kejing Wang, Zhi Zheng

Introduction and project goals

We found an interesting notion -- BookCrossing, which means "leaving a book in a public place (e.g. wooden hutches, park benches or abandoned phone booths) to be picked up and read by others, who then do likewise". This term is derived from the free website *bookcrossing.com*, which encourages the notion into practice by providing an online mechanism to track the books "released into the wild", and targets to "make the whole world a library".

With the robust dataset crawled from *bookcrossing.com* by Cai-Nicolas Ziegler that offers information of those books and ratings of readers whoever found them, all kinds of statistics and data backed up by real-world acts, we aimed to create a web application for book lovers that not only enables them to explore this collection of circulating books but also facilitates the search of books and offers user-specific recommendation based on personal taste.

Our web application *"Book4u"* has four general functionalities: a fact page for exploration, a search page to target books by tile/ authors/ publishers, a login system together with a bookshelf feature to save users' liked books, and a recommendation page based on the user's bookshelf content. The detailed list of features will be covered in the Architecture section.

Data Source and Data Preprocessing

Our group chose to use the existing data collection of three interrelated .csv files from one single source: https://www.kaggle.com/ruchi798/bookcrossing-dataset. We used python and pandas library to perform the data inspection and cleaning.

The "Books" file lists all aspects of information on books, including ISBN, the numeric commercial book identifier which is intended to be unique. There are about 280,000 rows and 8 attributes. This dataset contains about 102,00 different authors and 17,000 different publishers. In terms of data cleaning, we dropped rows that contained non-integer or unreasonable values for the numeric attribute "year of publication" (e.g. only keep rows with year in the range [0, 2020]).

The "Book_Ratings" file lists every rating as a row. There are about 1.1 million rows and 3 attributes (user, ISBN, rating). After an inspection, we deleted all ratings of 0, the default value when the rating is actually not given. The remaining ratings on a scale of 1 to 10 take up

430,000 rows, which is 37.7% of all. The remaining ratings have a mean of 7.6 with an std of 1.84, indicating a fair validity.

The “User” file lists the demographic information of each user. There are about 280,000 rows and 3 attributes (user, location, age). The quality of this data file was quite undesirable due to the unreliable user input of age and location. The "Age" column had only 40% non-empty inputs, with values ranging from 0 to 244. Same issues with Location, it had a large number of blanks and untrustful values. Thus we only kept the user column, which was referenced by the rating table. We then added the “email” and “password” columns on this file, making it a “user” table of our database, and set the two added columns to default value for these original users.

Finally, by conducting entity resolution, we made sure that the keys we chose for all our tables are unique.

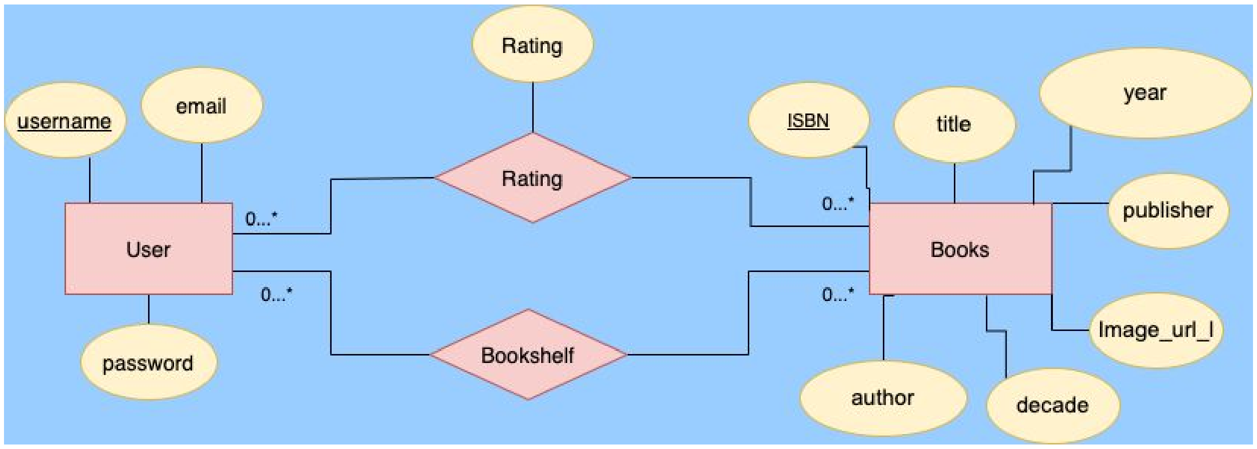
Relational Schema and ER Diagram

Users (<u>username</u>, email, password)

Books (<u>ISBN</u>, title, author, year, publisher, image_url_l, decade)

Ratings (<u>username</u>, <u>ISBN</u>, rating)

Bookshelf (<u>username</u>, <u>ISBN</u>)



This decomposition is in both 3NF and BCNF.

Description of system architecture

Below are the general descriptions and features of what each of our pages accomplishes (**<u>Note</u>**: queries for features are listed in *Appendix A*):

1. Dashboard Page:

On the top of the page, users can input the author, title, or publisher to the search bar and it will redirect to the search result page. Our dashboard page also provides two lists of books containing title, author, and cover to users. The list of books in the left column is the top 10 rating books of our database and on the right side is the top 3 rating books of each decade. Users can click the book cover in the dashboard to redirect to the *Subpage* of the book for detailed information.

Features implemented on this page:

- What are the highest-rated books over all years?
- What are the best books in each decade?
- Which are the books returned given the author?
- What are the books returned given a keyword of title?
- What are the books returned given a publisher?

2. Subpage:

The subpage pops up when the user clicks on any book cover on any page, including dashboard and search result pages, bookshelf page, and recommendation page. On the subpage, users can check detailed information about the book clicked before, including the book's ISBN, author, publisher, year of publication. Below the information box, is a slide show of other books written by the same author. In addition, there are three important user-interactive features on the subpage: A click on "buy this book" redirects users to the Amazon page of the book. Users can add the book to their own bookshelf. Users can submit their rating for the book by selecting the number of stars. Their operations to insert and update data will eventually be saved to our database.

Features implemented on this page:

- What is the ISBN, year of publication, publisher of the book?
- Where can users buy this book?
- What are other books written by the same author?
- How to add the book to the bookshelf? How to remove the book from the bookshelf?
- How to add ratings? How to update ratings?

3. Login Page:

Users can sign up, sign in, and log out on this page. Once logged in, the username will be passed to other pages, which is essential for the correct functioning of the Bookshelf page and Recommendation page.

Features implemented on this page:

- How to sign up? How to sign in?

4. Bookshelf Page:

After the user signed in, the bookshelf page is available for them to see the books they previously marked down. If the user wants to remove the book from their bookshelf, it can be done simply by clicking the book cover and clicking “remove from bookshelf” on the subpage.

Features implemented on this page:

- What are the books on the bookshelf of a given user?

5. Recommendation Page:

Once users add books to their bookshelf, they can see personalized recommendations based on the books on their bookshelf. In our recommendation query, we calculate the overlap of books between the current user's bookshelf and other users', and find the users with the most overlap. Then we return the books from those users' bookshelf that are not in the current user's bookshelf.

Features implemented on this page:

- What are recommended books for the current user?

Performance evaluation

Here, we list the optimization experiments of some queries (**Note**: All queries all listed in *Appendix A*). To perform optimization methods, we first used the “explain” clause in MySQL to display the query’s execution plan, and then experiment with our observations. We used “SET profiling = 1” and “show profiles” to record timings.

# Query1 (Appendix A.II.)

Get top 3 books per decade, books with less than 10 ratings excluded

```
 WITH bookRating AS (
   SELECT DISTINCT ISBN, AVG(rating) AS rating, Count(*)
   FROM Ratings
   GROUP BY ISBN
   Having Count(*)>10
   )
   , book_rating AS (
   SELECT b.ISBN, b.title, b.author, b.image_url_l, b.decade, r.rating
   FROM Books b JOIN ISBN_rating r ON b.ISBN = r.ISBN )
   , two_from_group AS (
   SELECT *, row_number() OVER (
                        PARTITION BY decade
```

```
                    ORDER BY rating DESC
            ) AS row_num
    FROM book_rating
    )
    SELECT ISBN, title, author, image_url_l, decade, Round(rating,2) AS rating
    FROM two_from_group
    WHERE row_num >= 1 AND row_num <=3
    ORDER BY decade DESC, rating DESC;
```

For the subqueries, the first returns ISBN and average book rating for the book with 10 or more ratings, the 2nd returns detailed book information including title, author, image_url_l, decade, rating for each book, and the 3rd returns the result of partition by decade and order by rating from the 2nd query. The final query simply returns the top3 rating books in each decade and order by decade and rating.

# Optimization 1: Push Down Projections (Failed)

The runtime before optimization: 0.2172s

The runtime after optimization: 0.2432s

```
WITH ISBN_rating AS (
    SELECT r.ISBN AS ISBN , r.rating AS rating
    FROM (SELECT DISTINCT ISBN, AVG(rating) AS rating, Count(rating)
          FROM Ratings
          GROUP BY ISBN
          Having Count(rating) > 10
         ) AS r
),
book_rating AS (
    SELECT b.ISBN, b.title, b.author, b.image_url_l, b.decade, r.rating
    FROM (SELECT ISBN, title, author, image_url_l, decade
          FROM Books) AS b JOIN ISBN_rating r ON b.ISBN = r.ISBN
),
two_from_group AS (
    SELECT *, row_number() OVER (
                    PARTITION BY decade
                    ORDER BY rating DESC
                    ) AS row_num
    FROM book_rating
)
SELECT ISBN, title, author, image_url_l, decade, Round(rating,2) AS rating
```

```
FROM two_from_group
WHERE row_num >= 1 AND row_num <=3
ORDER BY decade DESC, rating DESC
```

Here we tried to push down all projection actions before the join between Books and ISBN_rating, but this failed. Theoretically, pushing down projections to each table reduces the width of a table that goes into a join, but in this case, the projection projects over almost all attributes of both tables in the join, leaving the improvement too minor to counteract the time required for two additional projections on the tables.

# Optimization 2: Index on Ratings (ISBN, rating), btree -- (Succeeded)

The runtime before optimization: 0.2172s

The runtime after optimization: 0.0779 s

We create this two-attribute search key index on the table Rating with the hope that it can help with the “JOIN ON ISBN”, “GROUP BY ISBN” in the Aggregation and “GROUP BY ISBN, ORDER BY Rating” in the Partition method . Those operations all have a demand of either ordering of ISBN or ordering of rating or both. After the creation of the index, the query time reduces by 65%.

# Query2 (Appendix A.I.)

```
 WITH bookRating AS (
  SELECT DISTINCT ISBN, AVG(rating) AS rating, Count(*)
  FROM Ratings
  GROUP BY ISBN
  Having Count(*)>10
  )
  SELECT b.ISBN, b.title, b.author, b.image_url_l, Round(r.rating,2) AS rating
  FROM Books b JOIN bookRating r ON b.ISBN = r.ISBN
  ORDER BY r.rating DESC
  LIMIT 10;
```

The runtime before optimization: 0.27157775s

The runtime after optimization: 0.07548525s

The query is fairly simple in structure: join involved by ISBN, order the rating, and select the fields we need. The subquery returns the list of ISBN and average rating for books with10 or more ratings.

# Optimization 1: Failed

Add “r.rating > 9” to “ Having Count(ISBN)>10” to making table in the join smaller. The resulting join table should also be smaller, but the duration becomes larger. So we can know that the main problem here is not the size of the join table but should be the “full scan” problem. Let’s then try to create the index next. (0.29306300s)

# Optimization 2: Succeeded

We have 108,912 rows in our Rating table and based on the volume of our data, we can know that fully scanning is not optimal. Then try creating indexing on the "Ratings" table to make the full scan become the index scan and also optimize the order by clause. This optimization is successful with the duration 0.07548525s, which means the query time reduces by 75%.

# Query3 (Appendix A.III.)

```
WITH overlap AS (
   SELECT username, COUNT(ISBN) AS cnt
   FROM Bookshelf
   WHERE ISBN IN (
      SELECT ISBN
      FROM Bookshelf
      WHERE username = "${username}")
   AND username <> "${username}"
   GROUP BY username)
, similar_user AS(
   SELECT username
   FROM overlap
   ORDER BY cnt DESC
   LIMIT 10)
, rec_book AS(
   SELECT b.ISBN, b.username
   FROM Bookshelf b JOIN similar_user s ON b.username = s.username
   WHERE b.ISBN NOT IN(
      SELECT ISBN
      FROM Bookshelf
      WHERE username = "${username}"))
SELECT b.ISBN AS ISBN, b.title, b.author, b.image_url_l
FROM Books b JOIN rec_book r ON b.ISBN = r.ISBN;
```

The runtime before optimization: 4.137s

The runtime after optimization: 0.0019s

The query returns the list of books recommended to the user based on their bookshelf.

# Optimization 1: Succeeded

```
WITH user_bookshelf AS(
   SELECT ISBN
   FROM Bookshelf
   WHERE username = "${username}"
   )
, overlap AS(
   SELECT b.username, COUNT(b.ISBN) AS cnt
   FROM Bookshelf b LEFT JOIN user_bookshelf ub ON b.ISBN = ub.ISBN
   WHERE ub.ISBN IS NOT NULL AND b.username <> "${username}")
,similar_user AS(
   SELECT username
   FROM overlap
   ORDER BY cnt DESC
   LIMIT 10)
, rec_book AS(
   SELECT b.ISBN
   FROM Bookshelf b JOIN similar_user s ON b.username = s.username LEFT JOIN
   user_bookshelf ub ON b.ISBN = ub.ISBN
   WHERE ub.ISBN IS NULL)
   SELECT b.ISBN AS ISBN, b.title, b.author, b.image_url_l
   FROM Books b JOIN rec_book r ON b.ISBN = r.ISBN
```

Based on the number of rows in our Books table, it is obvious that a full scan is not optimal. So we create an additional secondary B+ tree index on attributes in the “Books” table to reduce scanning time. Then we found that using left join in each subquery to check the existence of attributes in other tables is highly efficient. The final query time is 0.0019s and is improved by 99%.

# Query4 (Appendix A.IV.)

```
SELECT title, author, publisher, image_url_l
FROM Books
WHERE title LIKE '%`+ inputBook + `%' OR author LIKE '%`+ inputBook + `%' OR publisher
LIKE '%`+ inputBook + `%’
LIMIT 20;
```

This query is for users to input some text (e.g. title, author, or publisher) and returns 20 books that are similar to the users’ input. We have already created the index for Books table, so here the performance is already improved largely. But we will try to improve more.

The runtime before optimization: 0.09617850s

The runtime after optimization: 0.07653825s

# Optimization 1: Succeeded

Sometimes using “OR” in our query will make the indexes created have no effect. So here we will optimize this by using “Union ALL” instead of “OR”. The runtime thereafter is 0.07653825s. And the modified query is shown as below:

```
SELECT title, author, publisher, image_url_l
FROM Books
WHERE title LIKE '%`+ inputBook + `%'
UNION ALL
SELECT title, author, publisher, image_url_l
FROM Books
WHERE author LIKE '%`+ inputBook + `%'
UNION ALL
SELECT title, author, publisher, image_url_l
FROM Books
WHERE publisher LIKE '%`+ inputBook + `%'
LIMIT 20;
```

Sometimes using “OR” in our query will make the indexes created have no effect. So here we will optimize this by using “Union ALL” instead of “OR”. The runtime thereafter is 0.07653825s and the performance is improved by 26%.

# Optimization 2: Failed

Clause LIKE “%text%” may also slow down the performance, let’s try to optimize the fuzzy search. Then we used POSITION('input' IN `field`) instead of LIKE clause, but the runtime is 0.08339000s. So we will stick with the LIKE clause.

Technical Challenges

One of the challenges we encountered is to clean our data. After we analyzed the statistics of the dataset, we found that not all of the columns are useful. Then we removed unnecessary columns (e.g. duplicate image links in Books table) and added important columns (e.g. password in the Users table). We then changed some data types to make the tables fit in our schema in MySQL and did the entity resolution to make sure all of our keys are unique.

Next, we faced difficulty in redirecting pages and passing values. We did a lot of research online to figure out the proper method for various cases: 1) Jump from one page to another, pass the value from the previous page to the latter; 2) Pass the value to the same page and refresh the page; 3) Jump to the website that out of our localhost; 4) Pass the value from one page to the remaining three without redirecting.

Also, we find it hard to cache each page in our website, which is to avoid refreshing in login page when we route to another page to store the user account.

Appendix A: *SQL Queries*

I. What are the highest rating books over all years?

```
 WITH bookRating AS (
  SELECT DISTINCT ISBN, AVG(rating) AS rating, Count(*)
  FROM Ratings
  GROUP BY ISBN
  Having Count(*)>10
  )
  SELECT b.ISBN, b.title, b.author, b.image_url_l, Round(r.rating,2) AS rating
  FROM Books b JOIN bookRating r ON b.ISBN = r.ISBN
  ORDER BY r.rating DESC
  LIMIT 10;
```

II. What are the best books in each decade?

```
WITH bookRating AS (
  SELECT DISTINCT ISBN, AVG(rating) AS rating, Count(*)
  FROM Ratings
  GROUP BY ISBN
  Having Count(*)>10
  )
  , book_rating AS (
  SELECT b.ISBN, b.title, b.author, b.image_url_l, b.decade, r.rating
  FROM Books b JOIN ISBN_rating r ON b.ISBN = r.ISBN )
  , two_from_group AS (
  SELECT *, row_number() OVER (
                PARTITION BY decade
                ORDER BY rating DESC
      ) AS row_num
  FROM book_rating
  )
  SELECT ISBN, title, author, image_url_l, decade, Round(rating,2) AS rating
  FROM two_from_group
```

```
   WHERE row_num >= 1 AND row_num <=3
   ORDER BY decade DESC, rating DESC;
```

III. What are the recommended books for the user?

```
WITH overlap AS (
   SELECT username, COUNT(ISBN) AS cnt
   FROM Bookshelf
   WHERE ISBN IN (
      SELECT ISBN
      FROM Bookshelf
      WHERE username = "${username}") AND username <> "${username}"
   GROUP BY username)
, similar_user AS(
   SELECT username
   FROM overlap
   ORDER BY cnt DESC
   LIMIT 10)
, rec_book AS(
   SELECT b.ISBN, b.username
   FROM Bookshelf b JOIN similar_user s ON b.username = s.username
   WHERE b.ISBN NOT IN(
      SELECT ISBN
      FROM Bookshelf
      WHERE username = "${username}"))
SELECT b.ISBN AS ISBN, b.title, b.author, b.image_url_l
FROM Books b JOIN rec_book r ON b.ISBN = r.ISBN;
```

IV. Which books are written by a certain author?/ What books' titles containing certain words?/ Which books are published by the publisher?

```
   SELECT title, author, publisher, image_url_l
   FROM Books
   WHERE title LIKE '%`+ inputBook + `%'
   UNION ALL
   SELECT title, author, publisher, image_url_l
   FROM Books
   WHERE author LIKE '%`+ inputBook + `%'
   UNION ALL
   SELECT title, author, publisher, image_url_l
   FROM Books
   WHERE publisher LIKE '%`+ inputBook + `%'
   LIMIT 20;
```

V. What are other books written by the same author?

```
 WITH curAuthor AS(
   SELECT author
   FROM Books
   WHERE ISBN = '${ISBN}'
  )
  SELECT b.ISBN AS ISBN, b.title AS title, b.image_url_l AS image_url_l
  FROM Books b JOIN curAuthor a ON b.author = a.author
  WHERE b.ISBN <> "${ISBN}"
```

VI. What is the ISBN, year of publication, publisher of the book?

```
SELECT title, author, publisher, year, image_url_l
FROM Books
WHERE ISBN =  '${inputISBN}';
```

VII. How to add the book to the bookshelf? How to remove the book from the bookshelf?

```
INSERT INTO Bookshelf (ISBN, username)
VALUES ("${ISBN}", "${username}");

DELETE FROM Bookshelf
WHERE username = "${username}" AND ISBN = "${ISBN}";
```

VIII. How to add ratings? How to update ratings?

```
INSERT INTO Ratings (ISBN, rating, username)
VALUES ("${ISBN}",${rating},"${username}");

UPDATE Ratings
SET rating = ${rating}
WHERE username = "${username}" AND ISBN = "${ISBN}";
```

IX. How to sign up? How to sign in?

```
INSERT INTO Users(username,email, password)
VALUES ("${username}",”${email}”,"${password}");

SELECT username, password
FROM Users
WHERE username = "${username}" AND password = "${password}";
```

X. What books are on the bookshelf?

```
SELECT b.ISBN as ISBN, b.title, b.author, b.image_url_l
FROM Bookshelf bs JOIN Books b ON bs.ISBN = b.ISBN
WHERE bs.username = '${username}'
```