**Amazon Book Review Webapp**

# **1. Introduction**

Nowadays there are numerous books, and people want to quickly find and read books they are interested in. The goal of the application is to provide a user-friendly website where readers can quickly search data about over 200,000 Amazon books. The data includes 1) basic information of each book like title, year, author and genre, 2) external links to Google Book which has more detailed metadata, and 3) reviews for each book with rating and comments.

Group members are:

| **Team Member** | **Name** | **Penn Email** | **Github username** |
| --- | --- | --- | --- |
| 1 | Zhen Zhang | zhennan@seas.upenn.edu | SeisSparrow |
| 2 | Xinyue Zhang | zxy@seas.upenn.edu | zxyupenn |
| 3 | Zhaoqin Wu | zhaoqinw@seas.upenn.edu | Chain1216 |
| 4 | Shuai Feng | fshuai@seas.upenn.edu | fshuai822 |

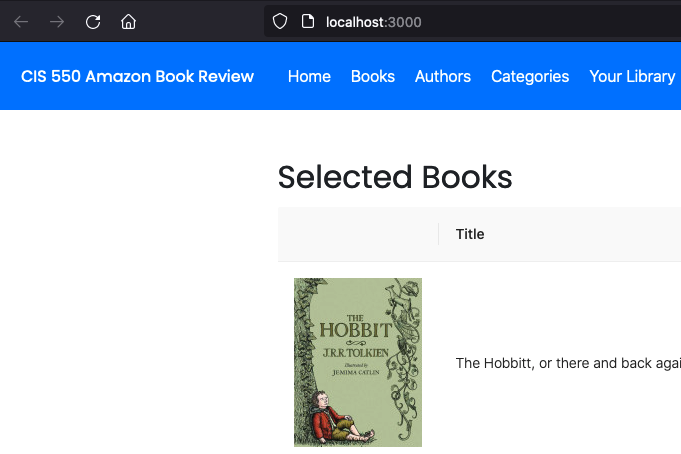
**Project Github Access:**

[*https://github.com/SeisSparrow/CIS5500GroupProject*](https://github.com/SeisSparrow/CIS5500GroupProject)

# **2. Architecture**

Data cleaning is done using python. SQL database is MySQL on Amazon RDS, with Datagrip software used to define the tables and import data.

The system is a responsive React Web Application for Amazon book review, which allows users to search the book, get detailed information about the book, and also get recommendations based on the relevant book author and genre. It has server side and client side, implemented in JavaScript. As shown by Figure 1, the web application includes 5 pages: ‘Home’, ‘Books’, ‘Authors’, ‘Categories’ and ‘Your Library‘, each of them contains information from the database mentioned below.



**Figure 1. Website Structure Illustration**

# **3. Data**

Our project leveraged 2 datasets:

## **3.1 Dataset - Books Details**

This dataset includes book details such as title, description, authors, publisher etc. for ~2MM unique books from Amazon

* Link: <https://www.kaggle.com/datasets/mohamedbakhet/amazon-books-reviews>
* Summary Statistics:
  + # of rows: 212,404 (We might use a subset of the data)
  + # of attributes: 10
  + Title: join key (unique id)
  + Description: 143,870 non-null records (68%)
  + Authors: 180,755 non-null records (85%)
  + Image: 160,250 non-null records (75%)
  + PreviewLink: 188,551 non-null records (89%)
  + Publisher: 133,064 non-null records (63%)
  + PublishedDate: 182,813 non-null records (86%)
  + InfoLink: 188,550 non-null records (89%)
  + Categories: 147,786 non-null records (70%)
  + RatingsCount: 49,752 non-null records (23%); Mean value: 21.25; Standard Deviation: 201.34; Min: 1; Max: 4895

## **3.2 Dataset - Books Rating**

This dataset includes book review details such as user id who rates the book, helpfulness of the review, rating score, full text of the review etc. from 3M users

* Link: <https://www.kaggle.com/datasets/mohamedbakhet/amazon-books-reviews>
* Summary Statistics:
  + # of rows: 3,000,000
  + # of attributes: 10
  + Id: 3MM non-null records
  + Title: 2,999,792 non-null records (100%)
  + Price: 481,171 non-null records (16%); Mean value: 21.76; Standard Deviation: 26.21; Min: 1; Max: 995
  + User\_id: 3MM non-null records (100%)
  + ProfileName: 2,432,297 non-null records (81%)
  + Review/helpfulness: 1,801,268 non-null records (60%)
  + Review/score: 3MM non-null records (100%); Mean value: 4.21; Standard Deviation: 1.20; Min: 1; Max: 5
  + Review/time: 3MM non-null records (100%)
  + Review/summary: 2,999,962 non-null records (100%)
  + Review/text: 2,999,992 non-null records (100%)

# **4. Database**

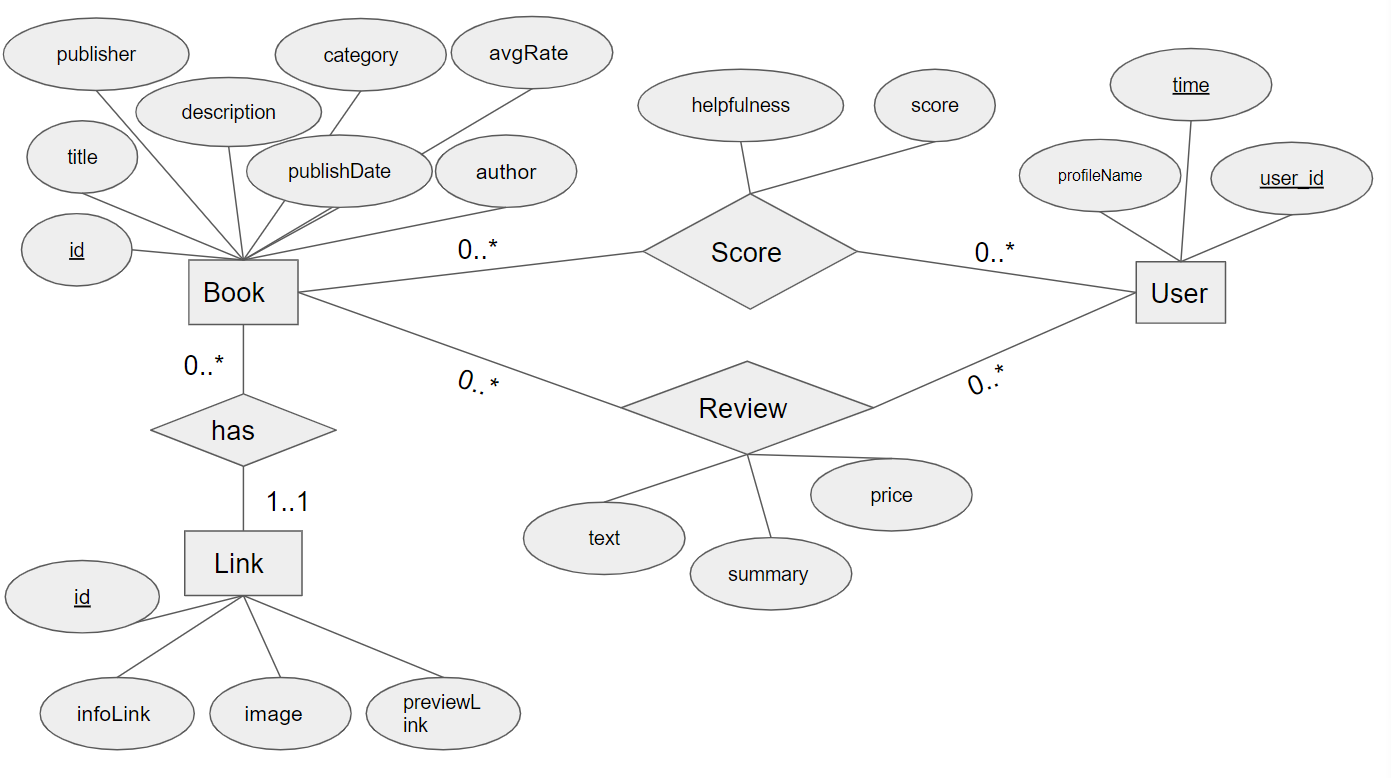
## **4.1 Data ingestion and cleaning**

Data pre-processing steps are as below.

* Read the two given csv into python and check for duplicated in both tables;
* Separated out id with key title from review table and remove the duplicates in to make sure each book has only one unique id so that we could use id as the unique key later in splitted tables;
* Filtered out user\_id null as it is one of the key attributes so it can’t be null;
* Removed the duplicated records for other attributes in review table (with key title);
* Merged the rest of review table with id and book table;
* Formatted the review/time attribute from Epoch to Date for readability;
* Splitted the published date to published year only to be consistent across all records;
* Stripped the brackets and quotes around categories and authors to show the text inside only;
* Splitted the dataset into 5 tables as we designed on ER diagram and exported into csv format for database import.

## **4.2 ER diagram**

Figure 2 shows the designed ER diagram. We can see there are 5 entities - Book, Link, Score, Review, and User.

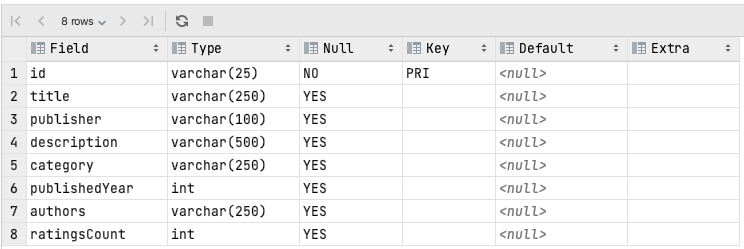
****

**Figure 2. ER diagram**

## **4.3 Table cardinality**

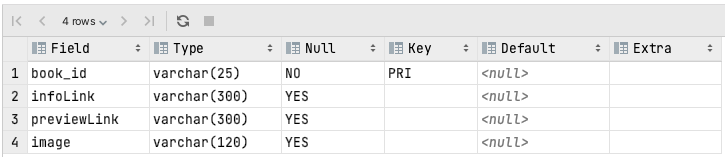
Corresponding to 5 entities shown in Figure 2, we adopted 5 tables. Below is the number of tuples for each table. The figures show the table schema.

* Book table: 13813



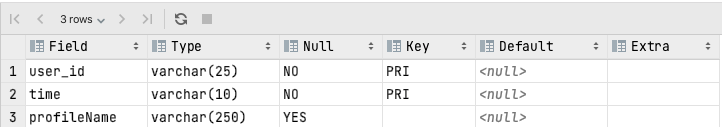
**Figure 3. Book table schema**

* Link table: 13813



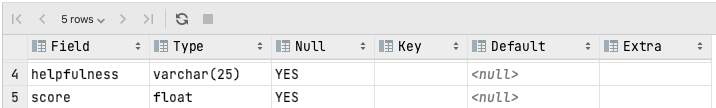
**Figure 4. Link table schema**

* User table: 295964



**Figure 5. User table schema**

* Score Table: 295964



**Figure 6. Score table schema**

* Review Table: 295879

## 

**Figure 7. Review table schema**

## **4.4 Normal Form and Justification**

* Book Table: BCNF; All 7 attributes (title, publisher, description, category, publishedYear, authors, ratingsCount) in Book table refer to the key ‘id’ (book id) only
* Link Table: BCNF; All 3 attributes (infoLink, previewLink, image) in Link table refer to the key ‘book\_id’ only
* User Table: BCNF; The 1 attribute (profileName) in User table refer to the whole keys ‘user\_id’ and ‘time’ only
* Score Table: BCNF; All 2 attributes (helpfulness, score) in Score table refer to the whole keys ‘book\_id’, ‘user\_id’ and ‘time’ only
* Review Table: BCNF; All 3 attributes (summary, text, price) in Review table refer to the whole keys ‘book\_id’, ‘user\_id’ and ‘time’ only

# **5. Web App description**

## **5.1 Home page**

Homepage will show a dozen of featured books from various genres, for example *The Hobbit,* *The Shawshank Redemption*, *The Art of War*, *A Brief History of Time*, etc. Each book has the cover page picture, and under the cover page is the title that has the hyperlink to Google Book. These featured books are selected based on the number of reviews and scores gained through review, only most scored ones will show in the Homepage.

## **5.2 Books page**

This page will show all books in the database, with book title, author, category, average score and cover image by joining the tables in the database, also it can direct you to a single book page with a hyperlink of each book to view the detailed information. Users can narrow down the range of books and search specific books by entering title or category. Also it achieves interaction between authors page and categories page by clicking the hyperlink. Additionally, with each single book page, it shows the brief introduction and selected reviews, also users can switch between the books page and single book page.

## **5.3 Authors page**

This page will show all books grouped by authors. Authors can be sorted according to different criteria, like name, number of published books, genre of most published books, etc.

## **5.4 Categories page**

This page shows top 10 book categories with most books, within which top 10 books that are mostly reviewed and best scored are displayed, sorted by number of reviews and average score. Info button is introduced for each book to jump to the book details on book page for users to further learn about each book; The save button is also introduced for users to save their favorite books into their own library on your library page.

## **5.5 Your Library page**

This page is mainly for users to view their favorite books that have been saved earlier when clicking on the save button. The login-in and sign-up pages are included for authentication purposes. It prompts users to access their own account to view the books they saved. Also, recommended books are displayed to introduce relevant books in regards to the users’ favorite books - the most popular books (mostly reviewed and best scored) are recommended to the users with the same authors or in the same categories as the favorite books.

# **6. API Specification**

## **6.1 Homepage**

sel\_books, top\_review

/books->

* sel\_books(req, res):

Route Functionality :

This route will return a table of top scored books, showing in details of book\_title, author\_name, category, published\_year, publisher and total\_scores. The record of each Book\_title is linked to the book page, and each author\_name is linked to the author page to show further details.

Request path :

app.get(‘/sbooks’, routes.sel\_book)

Request parameters:

None

Response parameters:

{int book\_title, string author\_name, string category, string published\_year, string publisher and int total\_scores} in result of JSON arrays

/review->

* top\_review(req, res):

Functionality:

This route will return a table of reviews for top scored books, showing in details of review\_content and book name.

Request path:

app.get(‘/review’, routes.top\_review))

Request parameters:

None

Response parameters:

{string book\_title, string review\_content} in result of JSON arrays

## **6.2 Authors page**

/author ->

Route functionality: Returns information about a author, specified by name

Request path: app.get('/author', routes.author)

Query Parameters: author (string)

Response parameters, types, and descriptions:

{ results: (JSON array of

{ authors (string),

books (string),

genres (string),

wikipediaLink (string)})}

/search/authors ->

Route functionality: Returns an array of selected attributes for authors that match the search query

Request path: app.get('/search/authors', routes.search\_authors)

Query Parameters: author (string)

Response parameters, types, and descriptions:

{ results: (JSON array of

{ Title (string),

Author (string),

Category (string),

Published year (string),

Publisher (string),

Number of reviews (int) })}

## **6.3 Books page**

* all\_books:

Route functionality:show all the books in the database, join all the tables to get the information of book cover, title, author, category, published year, publisher and average score.

Request path: app.get('/books', routes.all\_books)

Query Parameters: book (string)

Response parameters, types, and descriptions:

{ results: (JSON array of

{ Image(string),

Title (string),

Author (string),

Category (string),

Published year (string),

Publisher (string),

Average rating (int) })}

* book:

Route functionality: get detailed information about the specific book with book link, cover, title, author, published year, intro, average score and selected reviews.

Request path: app.get('/book', routes.book)

Query Parameters: book (string)

Response parameters, types, and descriptions:

{ results: (JSON array of

{ Image(string),

Title (string),

Author (string),

Category (string),

Published year (string),

Publisher (string),

Average rating (int) })}

* review:

Route functionality: get the reviews of the given book title as the request.

Request path: app.get('/review', routes.review)

Query Parameters: title (string)

Response parameters, types, and descriptions:

{ results: (JSON array of

{ Summary(string),

Text (string),

ProfileName (string),

Time (string) })}

* search\_books:

Route functionality: search books by specific book title in the database.

Request path: app.get('/search/books', routes.search\_books)

Query Parameters: book (string)

Response parameters, types, and descriptions:

{ results: (JSON array of

{ Image(string),

Title (string),

Author (string),

Category (string),

Published year (string),

InfoLink(string),

PreviewLink(string),

Average rating (int) })}

* search\_categories:

Route functionality: filter books by specific category.

Request path: app.get('/search/categories', routes.search\_categories)

Query Parameters: category (string)

Response parameters, types, and descriptions:

{ results: (JSON array of

{ Image(string),

Title (string),

Author (string),

Category (string),

Published year (string),

Publisher (string),

Average rating (int) })}

## **6.4 Categories page**

* cat\_top10:

Functionality: Get the top 10 books info for each category (one of the top 10 most popular categories).

Request path: app.get('/top\_categories/:category', routes.cat\_top10)

Request parameters: Category (string, required, book category)

Response parameters:

{ results: (JSON array of

{ Category(string, book type),

Id (string, book id),

Title (string, book title),

Authors (string, book author),

avg\_score (integer, book average review score),

cnt\_review (integer, total number of reviews),

Image (string, book image link)})}

* user\_fav\_book:

Functionality: Store user name (email) and book id into database when user clicked on the save button on category page.

Request path: app.post('/user\_fav\_book', routes.user\_fav\_book)

Request parameters: email (string, required, user email); id (string, required, book id)

Response parameters: None

## **6.5 Your Library page**

* user\_fav\_book: same route as on Category page.
* user\_fav\_book\_all:

Functionality: Get the book id and details for the user that is currently logged in.

Request path: app.get('/user\_fav\_book\_all/:email', routes.user\_fav\_book\_all)

Request parameters: email (string, required, user email)

Response parameters:

{ results: (JSON array of

{ email (string, user email),

id (string, book id) ,

title (string, book title) ,

image (string, book image link) })}

* user\_fav\_book\_recommend\_auth:

Functionality: Get the books with details that are recommended for the user that is currently logged in; The logic for recommendation is to select books with the same authors as the books that user saved.

Request path: app.get('/user\_fav\_book\_recommend\_auth/:email', routes.user\_fav\_book\_recommend\_auth)

Request parameters: email (string, required, user email)

Response parameters:

{ results: (JSON array of

{ id (string, book id) ,

avg\_score (integer, book average review score),

cnt\_review (integer, total number of reviews),

title (string, book title) ,

image (string, book image link) })}

* user\_fav\_book\_recommend\_cat:

Functionality: Get the books with details that are recommended for the user that is currently logged in; The logic for recommendation is to select books in the same categories as the books that user saved.

Request path: app.get('/user\_fav\_book\_recommend\_cat/:email', routes.user\_fav\_book\_recommend\_cat)

Request parameters: email (string, required, user email)

Response parameters:

{ results: (JSON array of

{ id (string, book id) ,

avg\_score (integer, book average review score),

cnt\_review (integer, total number of reviews),

title (string, book title) ,

image (string, book image link) })}

## **6.6 Login and Sign-up pages (Your Library Page Authentication)**

* user\_reg:

Functionality: Store token, username and password into user login table when user first registers on sign-up page

Request path: app.post('/user\_register', routes.user\_reg)

Request parameters: email (string, required, user email); password (string, required, book id); token (string, required, string of user email + password)

Response parameters: None

* user\_val:

Functionality: Get stored token from user login table to validate user prompt info (user + password)

Request path: app.get('/user\_val', routes.user\_val)

Request parameters: email (string, required, user email); password (string, required, book id)

Response parameters:

{ results: (JSON array of

{ email (string, string of user email + password) })}

# **7. Queries**

## **7.1 Example Query 1**

***/\* 1.*** Show reviews along with book images, scores etc. for books that are most popular with most reviews or highest score

**This Query stores intermediate tables to store most popular books in terms of highest score and with the most numbered reviews. Homepage will randomly select top reviews based on how they are associated with the two tables.**

**WITH highest\_score AS (**

**SELECT BOOK.id, BOOK.title,Link.image, SUM(Score.score) AS scores, COUNT(Review.text) num\_review**

**FROM BOOK**

**JOIN Link**

**ON BOOK.id = Link.book\_id**

**JOIN Score**

**ON BOOK.id = Score.book\_id**

**JOIN Review**

**ON Score.user\_id = Review.user\_id**

**GROUP BY BOOK.id, BOOK.title, Link.image**

**ORDER BY scores DESC**

**LIMIT 0,10**

**),**

**most\_review AS (**

**SELECT BOOK.id, BOOK.title,Link.image, SUM(Score.score) AS scores, COUNT(Review.text) num\_review**

**FROM BOOK**

**JOIN Link**

**ON BOOK.id = Link.book\_id**

**JOIN Score**

**ON BOOK.id = Score.book\_id**

**JOIN Review**

**ON Score.user\_id = Review.user\_id**

**GROUP BY BOOK.id, BOOK.title, Link.image**

**ORDER BY num\_review DESC**

**LIMIT 0,10**

**)**

**SELECT Review.text, BOOK.title, Link.image, SUM(Score.score) AS scores**

**FROM Review**

**JOIN Link**

**ON Review.book\_id = Link.book\_id**

**JOIN Score**

**ON Review.book\_id = Score.book\_id**

**JOIN BOOK**

**ON Review.book\_id = BOOK.id**

**WHERE Review.book\_id IN (SELECT id FROM highest\_score) OR Review.book\_id IN (SELECT id FROM most\_review)**

**GROUP BY Review.text, BOOK.title, Link.image;**

## **7.2 Example query 2**

***/\* 2. List out all the books in the database \*/***

***select b.id, b.title, b.category, b.publishedYear,b.publisher, b.description, b.authors, b.ratingsCount, l.image, l.infoLink, l.previewLink, s.averageScore***

***from BOOK b inner join Link l***

***on b.id = l.book\_id***

***inner join***

***(***

***select book\_id, format(round(avg(score),1), 1) as averageScore***

***from Score***

***group by book\_id***

***) s***

***on b.id = s.book\_id***

***order by ratingsCount desc***

## **7.3 Example query 3**

/\* 3.Select top 10 popular books of the specific category based on the number of reviews and average score of each book (Merge 3+ tables + Subquery + Aggregation)\*/

**with category\_info as**

**(select distinct a.category, a.id, *round*(*avg*(score),1) as avg\_score, *count*(\*) as cnt\_review**

**from BOOK a**

**inner join Score b**

**on a.id = b.book\_id**

**inner join Review c**

**on b.book\_id = c.book\_id**

**and b.user\_id = c.user\_id**

**and b.time = c.time**

**where a.category = '${category}'**

**group by a.category, a.id),**

**-- select top 10 books for each category based on score and review**

**category\_top10\_book as**

**(select \*, *row\_number*()**

**over(**

**partition by category**

**order by cnt\_review desc, avg\_score desc**

**) as rowNo**

**from category\_info)**

**select distinct a.category, b.id, b.title, b.authors, a.avg\_score, a.cnt\_review, c.image**

**from category\_top10\_book a**

**inner join BOOK b**

**on a.id = b.id**

**inner join Link c**

**on b.id = c.book\_id**

**where a.rowNo <= 10**

**order by a.category, cnt\_review desc, avg\_score desc**

## **7.4 Example Query 4**

***/\* 4.Record books that user saves and show recommended books that are related to the book that the user selects with the same author and genre with top scores (Merge 3+ tables + Subquery + Aggregation)\*/***

***with User\_fav\_details as***

***(select a.\*, b.title, b.authors, b.category***

***from User\_fav a***

***inner join BOOK b***

***on a.book\_id = b.id***

***where email =* '${email}'*,***

***same\_authors as***

***(select a.id, avg(score) as avg\_score, count(\*) as cnt\_review***

***from BOOK a***

***inner join Score b***

***on a.id = b.book\_id***

***inner join Review c***

***on b.book\_id = c.book\_id***

***and b.user\_id = c.user\_id***

***and b.time = c.time***

***inner join User\_fav\_details d***

***on a.authors = d.authors***

***and a.id <> d.book\_id***

***group by a.id***

***order by cnt\_review desc, avg\_score desc LIMIT 10),***

***same\_catg as***

***(select a.id, avg(score) as avg\_score, count(\*) as cnt\_review***

***from BOOK a***

***inner join Score b***

***on a.id = b.book\_id***

***inner join Review c***

***on b.book\_id = c.book\_id***

***and b.user\_id = c.user\_id***

***and b.time = c.time***

***inner join User\_fav\_details d***

***on a.category = d.category***

***and a.id <> d.book\_id***

***group by a.id***

***order by cnt\_review desc, avg\_score desc LIMIT 10)***

***-- Choose top 10 from each table***

***(select distinct a.\*, b.title, c.image***

***from same\_authors a***

***inner join BOOK b***

***on a.id = b.id***

***inner join Link c***

***on b.id = c.book\_id)***

***union***

***(select distinct a.\*, b.title, c.image***

***from same\_catg a***

***inner join BOOK b***

***on a.id = b.id***

***inner join Link c***

***on b.id = c.book\_id);***

## **7.5 Example Query 5**

/\* 5. Show a filter to allow the user to search books by author, which shows the book information including title, year, publisher etc as well as google book link. \*/

**SELECT *\****

**FROM BOOK JOIN Link ON id = book\_id**

**WHERE authors LIKE '%${req.query.author}%'**

**AND authors NOT LIKE '%,%'**

**AND authors NOT LIKE '%"%'**

**AND authors NOT LIKE '.%'**

**AND authors NOT LIKE ' %'**

**ORDER BY authors**

**LIMIT 100**

# **8. Performance evaluation**

Before optimization, it used to take 3-5 min to load the contents on the website, which is not a good experience for users. So we optimized the query to improve the performance of loading data, for example, since we have various attributes in our tables, we did the selection first to filter the attributes we need and then joining the tables so that we would be able to reduce the size of the intermediate results and minimize the cost of evaluating the expression; also there are several tuples which includes special symbol in it, so we use “where” to filter and limit the size of the output. Additionally, for the huge query results which included ~200K tuples, we put a limit on it to reduce the number of tuples in the results.

Other than query, we also optimized the user log-in feature; it used to rely on interacting with the database to identify the username that is currently logged-in, which takes a lot of time for the client-server interaction. Later we changed to using sessionstorage, which is much faster and stable to keep the log-in status up-to-date.

The optimization largely improved the performance - it is taking <20 seconds to show all contents now.

# **9. Technical challenges**

We found out that we cannot run npm start on the client side every time we pull the latest version of the project after team members merged their local branch to the main branch. By studying the code, we find that node\_modules inside client documents vary in cache every time team members run the code in their host computer. The inconsistent caching later being pushed to github will lead to the client npm start error once team members pull the whole program. We come up with a set up rule every time we pull the latest main branch as:

1. Delete the whole node\_modules and package-lock.jason file inside Client

2. Run npm cache clean -f

3. Run npm install –force

4. Run npm start