CS 411 Project Report - Team Question

## Project Outline

Our project was to build a book recommendation database, centered around a user’s ability to enter reviews. Based on how the user rated certain books, we would update recommendations on what to read next. Our original plan was to use two separate datasets on Kaggle and join them together to form our main Books and Reviews table. However, we quickly realized that this would be quite difficult, as the two datasets had different columns, which would make combining them complicated. We would need to sanitize one dataset, prune the columns that only existed in one of the two datasets (which would cause us to lose information), and then load the reduced datasets into MySQL. In the end, we decided to use the most extensive Kaggle dataset and find a dataset with author information to get additional detail. This solution still required us to join on a string column and didn’t work for every book, but it allowed us to determine the popularity of most authors.

## Implementation

Our final database implementation has the following tables: Users, Ratings, Books, Authors, Friends and Publishers. In our original ER diagram, we didn’t plan on adding a Friends table, but we decided that having an additional option for customization would allow the user to have greater flexibility and more options for book recommendations. We also decided to add a Popularity column in the Authors table, again to give us more options when recommending. Our advanced queries consist of several stored procedures that take reviews from users with similar reading lists, books from the same authors or publishers, or from a user’s friends. These programs allow us to customize which books are shown to the user. In addition to this, we added an option to pick what kind of rating (out of 10) is required for a book to be considered “good” and for books to be recommended based on it. There’s also an option (only available for “recommend from similar users”) that allows you to pick the number of users with similar reading lists who read the book for it to be recommended. We did not include any indices in our database, as all of the indexing schemes we tried (even for the later stored procedures in Stage 5, where we added an index in Books on Authors) were either tied with the original query speed, or even slower than using a full database scan in some cases.

## Interface

The OpenSourceBooks project has two different interfaces. The first is a web server, hosted on GCP. This application includes several pages. Once you login with your username an password, you can look at your friends (and add/delete from your friends list), browse books from the full catalog to search for new recommendations, view or add/edit/delete your reviews, and view book recommendations given your current reading list and ratings. The web server also has an admin portal. When implementing the web server initially, we realized that it was vulnerable to a SQL injection attack (the user could enter arbitrary SQL code into a textbox, and our server would then execute it). Once we realized this, we decided to implement several features to increase the security of our site. The first step was to sanitize user input to make such an attack impossible. Next, we wanted to guard against automated bots “review-bombing” certain books, ie posting large numbers of positive or negative reviews for certain books and affecting their popularity. As a compromise, we built a trigger when inserting reviews, as well as a stored procedure that can only be triggered by logging into the web server’s admin portal. Once an admin logs in, they can call a procedure to update all authors’ popularity based on the book reviews. This procedure also detects suspicious jumps in an author’s popularity and doesn’t update if the author’s popularity increased or decreased by too much.

The second way to access the OpenSourceBooks database is through the Discord integration. We built a Discord application using a bot that allows you to view reviews or browse books. Originally, we planned to make this fully integrated with the database (like the website), but we quickly realized that this would require a user to post their password in plaintext on Discord, which was not acceptable to us. Due to this, we were forced to scale back our Discord application, and make it only have features that are available to users who do not need to log in. If we had more time, we could have found a way around this, allowing us to expand the Discord application by implementing its full functionality.

## Division of Work

The way we decided to split the work was quite different from our original proposal. Instead of having a single member be responsible for a large component such as the SQL database, the Web server, etc. we decided to work together on each component. Instead, each of us worked on a different aspect of each component, to allow us to work in parallel. For example, when we developed the advanced queries, each of us worked on a different query separately, and we then designed the full recommendation algorithm (RecommendFromAll procedure) once each of us had completed the previous stored procedures. Overall, this division of work greatly increased our workflow and allowed us to be more productive.

## Closing Notes

Our final database implementation is quite similar to our original proposal, with the main changes being the use of only a single base dataset (rather than the two we originally planned for), and the addition of several security features in our application. In the future, we could improve the Discord interface, but our main idea for improvement is to add better recommendations. In particular, implementing machine learning algorithms to produce recommendations rather than generating them from a limited set of criteria would probably produce more relevant results for the users on OpenSourceBooks. Overall, the database has met its design goals and has gone beyond. We are able to produce useful recommendations for what to read next, and the website has more integration and security features than originally planned. We are quite happy with what we were able to make this semester.