For this stage, your team is expected to submit the final project deliverables, including a project reflection report and a video demo.

Your project reflection report should contain the following topics. The report would be graded on completeness and correctness.

1. ~~Please list out changes in the directions of your project if the final project is different from your original proposal (based on your stage 1 proposal submission).~~
    a. ~~Aaron~~
2. ~~Discuss what you think your application achieved or failed to achieve regarding its usefulness.~~
    a. ~~Aaron~~
3. ~~Discuss if you changed the schema or source of the data for your application~~
    a. ~~Aaron~~
4. ~~Discuss what you change to your ER diagram and/or your table implementations. What are some differences between the original design and the final design? Why? What do you think is a more suitable design?~~
    a. ~~Aaro~~n
5. ~~Discuss what functionalities you added or removed. Why?~~
    a. ~~Raymond~~
6. ~~Explain how you think your advanced database programs complement your application.~~
    a. ~~Raymond~~
7. **Each team member should describe one technical challenge that the team encountered. This should be sufficiently detailed such that another future team could use this as helpful advice if they were to start a similar project or where to maintain your project.**
    a. **~~Aaron, Leon, Raymond~~, Pinakin**
8. **Are there other things that changed comparing the final application with the original proposal?**
    a. **Pinakin**
9. **Describe future work that you think, other than the interface, that the application can improve on.**
    a. **Pinakin**
10. **Describe the final division of labor and how well you managed teamwork.**
    a. **Leon**

Video:

The video should be between 2-5 minutes long (7 minutes is the absolute max). Make it fun. Think of it as your chance to advertise your project to investors who find it promising.

Project reflection report:

1. **Please list out changes in the directions of your project if the final project is different from your original proposal (based on your stage 1 proposal submission).**
2. **Discuss what you think your application achieved or failed to achieve regarding its usefulness.**
3. **Discuss if you changed the schema or source of the data for your application**
4. **Discuss what you change to your ER diagram and/or your table implementations. What are some differences between the original design and the final design? Why? What do you think is a more suitable design?**

Overall, we managed to keep the direction of the project consistent with our original proposal. We proposed a Steam clone website that would allow users to browse games under a simple, minimalistic user interface as an alternative to the cluttered website from Steam. In the end, this is what our project achieved.

Most of the features that we promised to deliver, such as browsing games based on criteria and bookmarking games, are implemented. Understandably, there are some features that ended up being too ambitious, like viewing full information for every game like videos and reviews, and were not implemented due to time constraints. Further, the user interface in the mockup within the original project was way more polished, which was ambitious to implement as we did not learn much frontend design in this class to be able to execute on that plan.

To make up for this game information feature, we added a discounting system, which allows for users to apply discount codes to certain games satisfying criterion on game name, genre, and release date. This was also used to demonstrate our understanding of stored procedures and triggers. This discounting system allows applying a particular discount on Games satisfying the provided criterion, and prevents the case where the price becomes more expensive if the game is bookmarked by the user.

Another change we did was the criteria for searching on Games was done only by name. This is because we realized that we added two more ways of searching on Games that were based on a variety of parameters like website, genre, price, number of players, and minimum and recommended requirements to satisfy the advanced query requirements, and so there would be too many redundant ways to perform a query on Games.

The source of the data was kept the same, which was from one of the TA suggested datasets. For the schema of the database, we made a few changes from the ER diagram proposed in the stage 1 project report. Originally, we intended to store information about the minimum and recommended requirements for games as a relationship between games and requirements. However, we realized, with the TA's advice, that a minimum requirement is likely to correspond to the same recommended requirement, and that there is a strong correlation between requirements among different platforms. This means that it would make more sense to store all minimum and recommended requirements for each platform within a single table, which is how the final implementation is done.

Another change we added was an images URL column to the Games table. This allows us to store an image URL representing the game cover photo for each game, making the final website look much more visually appealing than simply having a wall of text describing each game by their name, short and long description, and website link. We believe that this delivers on the vision of the project to provide a less cluttered version of Steam to browse games.

5. **Discuss what functionalities you added or removed. Why?**
   We have added several functionalities to our relational database-centric web-based application, which is built using a Steam games dataset, Svelte for the frontend, and Java for the backend. The functionalities added include:
   - User registration and login:
     - This allows users to create an account with a username, name, and email, and log in to access personalized features.
   - Updating username:
     - Users have the option to change their username on the website, providing flexibility and enhancing user experience.
   - Bookmarking games:
     - Users can bookmark their favorite games, enabling a personalized experience tailored to their preferences.
   - Game recommendations:
     - Based on users' bookmarks, the application provides personalized game recommendations.
   - Specialized game search:
     - Users can search for games using two specialized queries to find games based on specific criteria.
   - Discount application:
     - Users can apply a discount to games based on genre, release date, and game name, and view the old and new prices.

   No functionalities were removed in the process, as all added features contribute to enhancing the user experience and providing valuable services to users.

6. **Explain how you think your advanced database programs complement your application.**
   The advanced database programs implemented in this application complement the application in several ways:

- Efficient data management:
    - By using a relational (SQL) database stored in GCP, the application can manage large volumes of data efficiently, ensuring that users can access and search the Steam games dataset with ease.
- Personalized user experience:
    - The database programs enable users to bookmark games and receive personalized recommendations based on their preferences, which enhances user engagement and satisfaction.
- Advanced search capabilities:
    - The specialized queries available for searching games allow users to find games that meet specific criteria, making it easier for them to discover new and interesting titles.
- Dynamic pricing:
    - The ability to apply discounts to games based on various attributes showcases the flexibility and adaptability of the database programs, offering users an incentive to explore more games and genres.

In conclusion, the added functionalities and advanced database programs contribute to a more user-friendly, personalized, and efficient application, making it a valuable tool for discovering and enjoying Steam games.

7. Each team member should describe one technical challenge that the team encountered. This should be sufficiently detailed such that another future team could use this as helpful advice if they were to start a similar project or where to maintain your project.

Aaron:

One technical challenge that the team encountered was implementing the bookmarking system. Coming up with the queries to query bookmarked games for a particular user, or add and delete games from the bookmarked list was straightforward, as this was similar to the SQL queries both discussed in lecture and solved in homework problems. However, it was difficult to figure out how to handle this on the frontend side. Specifically, to query / add / delete from the Bookmarks table, one must know the user name as well as the game name, as these are the keys for the Bookmarks table, which themselves are foreign keys to the Users and Games table, respectively. These are needed in the queries we designed to update the Bookmarks table.

So, I needed a way to determine 1. The username of the logged in user, if there is a logged in user, and 2. the game name selected by the user. This was difficult to implement in the frontend as I did not have much frontend knowledge, and so it was extremely frustrating to understand how to pass a simple variable of what the user selected between different components of the frontend. I spent quite some time trying to figure out how to share the logged in user variable between the different components. I first started by attempting to pass in the variable itself, but the value would not update if another component updates it. So, the solution was to have a reference to a variable outside of the current component, by using the export keyword. Then, to figure out what game the user is selecting, I added a button on each game displayed by the frontend. Because of my lack of frontend knowledge again, it took quite some time to figure out how to determine the game name corresponding to each button. I was able to achieve this by grabbing the data from the for each loop that iterates over all the queried games results, since each button is rendered for every iteration in this loop, and so there is a one to one correspondence towards games. Finally, the last issue, which took the longest, was to figure out how to let the users know that the game was bookmarked. First, I created a component that displays the bookmarked games of the current user. When the user removes a bookmark, I was able to automatically refresh this view as the button to remove bookmarks is within the same component. However, the issue was that adding bookmarks happens within the games query component. I spent massive amounts of time trying to figure out how to force the bookmarks component to rerender automatically upon a bookmark add. However, in the end I realized this was going to take too much time for my frontend skills, and decided to simply add a button within the bookmarks viewer component itself to refresh the bookmarks, which the user can toggle to view their updated bookmarks list.

Raymond:

One notable technical challenge we encountered during the development of our relational database-centric web-based application was implementing the "apply discount" feature. This feature allows users to apply a discount to games based on genre, release date, and game name and view the old and new prices.

The primary challenges we faced during the implementation of this feature can be summarized into four main aspects. First, efficiently querying the dataset was a challenge due to the large size of the Steam games dataset. To tackle this, we carefully designed the database schema and fine-tuned the query structure to ensure efficient retrieval of relevant data. Investing time in researching and optimizing SQL queries helped us minimize the response time when users apply a discount based on their chosen criteria.

Second, ensuring accurate discount calculations was crucial for the proper functioning of the feature. We had to pay close attention to the data types used in our calculations and round the results to make sure that the displayed prices were accurate and properly formatted. Rigorous testing was performed to verify that the discount calculations were reliable and precise.

Third, handling edge cases was another challenge. When applying discounts, we had to consider various scenarios, such as games with prices below a certain threshold or games that were already discounted. To manage these situations gracefully, we developed a set of rules and validation checks. Implementing validation checks and exception handling in our Java backend helped us ensure a smooth user experience and maintain the feature's integrity.

Lastly, updating the frontend to display both the original and discounted prices after applying a discount required careful consideration. This involved modifying the Svelte components and ensuring that the new information was presented in a clear and visually appealing manner. We iterated on the frontend design, incorporating user feedback and usability best practices to guarantee that the discounted prices were displayed effectively.

In conclusion, by sharing these insights, we hope that future teams working on similar projects or maintaining our project will have a better understanding of the challenges related to implementing a discount feature and the steps taken to overcome them. By addressing these challenges, we were able to develop a robust and user-friendly discount feature that adds value to our application.

Leon:

Something that was pretty annoying when bringing database data from the backend routes to the frontend was that I didn't name things very well, so I had a bunch of errors related to incorrect routes and urls, because the fetch requests I made on the frontend components were to the wrong route or a nonexistent one. This is a really small but frequent error I made because my naming conventions were bad. For example, for the sign-in component, I had name mismatches throughout my backend and frontend because I defined the route called 'username' but was trying to fetch 'users' on the frontend form. So I had to go back and rename everything which added a lot of unnecessary work. Make sure to name your variables and routes properly and tell your teammates what conventions you are using.

Also, in general, it can be hard to diagnose errors during this stage where you are bringing your frontend and backend together. Your best friend is Inspect Element. When you run on localhost, use Inspect Element and check the Console and Network tabs to see whether your frontend is making the right calls to the backend. The Elements tab is also incredibly useful for diagnosing HTML errors and CSS styling errors for the future project member who is working on the frontend.

Finally, a great README.md file is extremely important. You should be updating this as frequently as possible, because if you don't, everyone except you will NOT know how to run your app.

Pinakin:

One challenge that our team faced during the development of our project was during the implementation of our indexing. Indexing is extremely important for the speed of our application, and without it, we would be forced to deal with a slower response time for the crucial SQL queries that made our application possible. However, there are different types of indexing techniques for each type of query. These techniques, specifically using a B-tree and hash indexing, each have their own advantages and disadvantages. We had to also consider the trade-offs between indexing and storage space. While indexing increases the speed of our queries, it also can increase the storage space required for the database. So, if we wanted to scale our application to include the entirety of the steam library, we would have issues and would need to evaluate the tradeoffs appropriately. We also needed to check whether the indexing we needed aligned with the type of query we had. Additionally, to create our indexes, we first had to look at which part of our query served as the main contributor towards the cost of the query. From there, we could evaluate whether or not the tradeoffs of the index's memory consumption was worth it. My advice to users attempting to recreate our project would be to truly evaluate the scale and the scope of what the application should be before implementing it. Do you want to have a social network aspect? Do you want the database to continuously update? All of these factors should be considered when designing the backend of the application.

8. Are there other things that changed comparing the final application with the original proposal?

Like most projects, the final result was somewhat different than what we planned. As mentioned by Aaron in the first response, our frontend application was simpler due to the scope of the project, and we ended up not implementing the ability to view the full information for every game. Instead, we were able to fully implement the bookmarking system and add a game recommendation system, which wasn't planned in the original project proposal. Additionally, in the initial project proposal, we intended on having a friends system within our application. This implied having a friends relation in our database design, and allowing users to view their friends on their designated profile page. For our final result however, we chose to not have a friends system in order to better focus on what our application was meant to be: a clean, straightforward, and intuitive alternative to the steam client that would allow users to view and learn about new games, along with their information effectively. Because of the time restraint, and because of the focus and requirements of our application, we decided to instead implement a discounting system. Our original proposal also mentioned a profile page. As our team developed the project, we realized that a profile page wasn't necessary for a viable product.

9. Describe future work that you think, other than the interface, that the application can improve on.

Other than the interface, future work for the application would include allowing users to see gameplay videos and reviews, which was part of our original proposal. Since our application is meant for users to explore and learn about as many games as they can, this would be a beneficial feature to have. User reviews allow users to have more information about a game from those who have played it, which can help them make an informed decision on whether or not the game is worth buying. In the future, the application could provide analytics for users to see their gaming history, such as the number of hours played or the genres they tend to play the most, but it's important to note that we would need to also collect that data per user for this feature to work as intended. Additionally, if we wanted to switch to displaying real-time data for the number of users currently playing the game, we could implement a stored procedure or we could use a tool like Apache Kafka or Apache Spark that would allow us to do so.

The application currently focuses on providing a clean and intuitive interface for browsing games on Steam. However, it would be interesting to integrate other platforms into the application, such as Xbox, PlayStation, or Nintendo. This integration would enable users to browse games from multiple platforms and have a centralized location to discover new games. Once again though, it's important to note that we would need to have a way to pull data from these platforms, either through an API or an existing database that's currently online.

The application could add analytics features that allow game developers to see how users are interacting with their games. By providing game developers with information on how users play their games, such as the number of hours played or how far users have progressed in the game, developers can improve their games to meet user expectations. Additionally, the application could provide analytics for users to see their gaming history, such as the number of hours played or the genres they tend to play the most.

Another improvement that could be applied to the application would be ORMs to abstract away the SQL to make it easier to understand and more efficient to manage data. However, obviously ORMs have their disadvantages like increased overhead resulting in slower performance and limited flexibility (and we were required to use SQL statements for the project). But it does reduce complexity of the codebase and can also protect from SQL injection attacks.

Since we're using Spring Boot, a Java framework for our backend, an interesting alternative to look into is jOOQ, which is a library that lets you write type-safe SQL statements in the backend, allowing you to have the fine-grained control of SQL in the backend (which ORMs may abstract away) but also maintaining security.

10. Describe the final division of labor and how well you managed teamwork.

The final division of labor was similar to the division on the original proposal. Leon handled the frontend components, while Raymond, Pinakin, and Aaron handled the backend components.

We managed teamwork beautifully, as we were able to deliver on the requirements. On every stage, we met in-person to divide each stage requirement and then plan out internal deadlines. Because of this, we were able to work together to complete the project requirements.