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).

When compared to our original stage 1 project proposal, our final project had a couple of significant changes. Instead of focusing on building a centralized online platform where users can play multiple games, we decided to narrow our scope for the sake of time. While we maintained the core vision of creating a centralized platform that allows users to interact, create their own rooms, and view user-related statistics, we temporarily excluded the game-playing functionalities to focus on developing the core database/backend functionalities instead. We still plan to incorporate the game-playing features in the near future.

Discuss what you think your application achieved or failed to achieve regarding its usefulness.

In regards to our application, we were able to successfully achieve a fully functional web application that is able to do all of the basic CRUD operations. In addition, we were able to successfully implement all of our SQL queries, as well as being able to do every non-game related functionality, including sending transactions, creating/deleting rooms, logging in and user authentication, and being able to view user statistics with mock/auto-generated data. This made it so that users could actually work with their data. While we were able to implement all of the core database functionalities, we were unable to complete the core-game functionality, which made it less useful for users who actually wanted to play the game.

Discuss if you changed the schema or source of the data for your application
The source of data for our application had to be changed due to our initial source not meeting the project requirements. Initially, we decided on using a Kaggle sourced dataset which acted more as a datacard. This datacard functioned as a game history log where each line denoted an action that occurred in the game sequentially. Our plan was that we would read this data in to somehow “simulate” a game, because the nature of our project was such that we have to generate data to have data. We can’t just source data and run a game that comes from users playing it. Therefore, to meet project requirements, we sourced another Kaggle dataset in csv format which represented “simulated” poker games with relevant data such as buy in, game results, balance, and more. We additionally decided to use an auto generated dataset to represent out of game functionality, such as user related data, transactions, shop, and more. Through this, we were able to get data for both parts of our application for the in-game and out-game functionalities. The ability to use auto-generated data for the second half was very helpful due to the fact that finding datasets for that specific data requirement was incredibly difficult.

The schema stayed the same. The table/table relations designed were not much different to what we ended up implementing due to the fact that they were designed upon the datasets we re-selected.

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?

We didn't end up making any changes to our original ER diagram/table implementations. We were able to avoid doing this by thoroughly planning out the design of the database ahead of time, as well as being intentional in making sure each table would work within the flow of the online platform.

Discuss what functionalities you added or removed. Why?

As stated before, we ended up removing the core-game functionalities in the interest of focusing on the database/backend functionalities of the website. We didn't end up adding any new functionalities, as the website was already complex and we didn't want to overcomplicate the experience for the user.

Explain how you think your advanced database programs complement your application.

We utilized advanced database programs such as triggers, stored procedures, aggregation, joins, and more to complement the features of our application. For example, our transactions query allowed us to successfully allocate every single transaction(whether that be through games non-game activities) for whatever user was logged in. Another query was the advanced stats query, allowing the user to view every individual move and the success rate of each move, which could be useful in future games they played. These advanced programs allowed us to ultimately have many stats/functionalities that could be easily accessible for the user.

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.

Kaushal: The biggest issue for me was running into connectivity issues with regards to the APIs and data stored on our GCP VM. I was regularly running into 403: Forbidden errors due to CORS policy. To circumvent this, documentation (Flask + CORS in our case) and resources such as stack overflow are very useful to understand what is going on (e.g. there may be a firewall preventing access, etc.). Additionally, I was having issues remotely connecting to the database stored on the GCP VM. I was able to ssh and access the database, but was not able to access it outside this shell. I was able to determine that setting the GCP VM such that all IP addresses can reach the database was a workaround to this issue. Lastly, a big issue initially was not understanding how to split up pages/functionality by components. I initially stored all the data in a singular component, but quickly realized that having a single component front end was not the optimal architecture and that splitting components based on the pages/functionality we needed makes the codebase much more maintainable and allows for easier feature integration.

Julia: The biggest issue for me was implementing Google OAuthentication using Angular. I had originally thought that one button to sign in with google would only require one or two files. As I tried implementing it myself, I realized that nothing was connecting. I tried watching tutorials, but each implemented it a bit differently than I wanted. I thought I could try to mix and match functions from different tutorials; however since I hadn't fully understood what I was doing, it still didn't work. I used the help of ChatGPT to explain what each part did, so that I could then rewrite it to do exactly what I want. Although people say to not use ChatGPT, I think it's really helpful when used in combination with other resources.

Ethan: The biggest issue for me was getting the mySQL database locally. We were originally using the GCP Cloud, but often it would shut down randomly and we would have to wait for it to reboot to continue testing. Midway through, we decided to move it locally which means install and set everything up. Initially, I tried to follow the same steps I did for the GCP Cloud, but many things were erroring. When I got it set up, I realized that I couldn't import CSV files. After lots of searching, I realized I needed to edit a file in the /etc folder. You had to edit the file through the terminal to change permissions. I got many errors like foreign keys failing and I didn't know why. It took a while, but I realized that the command to import in GCP was different from locally because the NULL value was imported as a string "NULL" instead of the actual value. So we had to add the line SET <attr> = NULLIF(<attr>, 'NULL');

Daniel: The biggest issue for me was implementing the transaction tables, specifically in how the tables were showing up in the UI. I was specifically having issues with the styling of the tables, as they were taking up the entire page(making them unreadable) and were not responsive to any styling edits made to them. Despite trying different general tips, I was unable to fix the error quickly. After looking through many stack-overflow pages, I was able to finally find the issue and was able to resolve it by utilizing ng:deep and another parent container to fit the table in. Ultimately, I learned the value of being resourceful.

Are there other things that changed comparing the final application with the original proposal?
Describe future work that you think, other than the interface, that the application can improve on

With regards to the concept, there wasn't much that changed with the final application and the original proposal except the following. Initially, we wanted to have the application as a game suite that hosted multiple games, but due to the complexity of this idea, we narrowed our scope down to focusing on one card game. In the end, we weren't able to get to the implementation of the game itself, because we were more focused on getting the other requirements satisfied (e.g. implementing all CRUD operations, advanced queries, etc.). The application can improve by integrating additional key features. As mentioned before, for our application to actually be considered a game suite, we should be able to implement multiple card games. Secondly, implementing some sort of ranking functionality, in which one can see their ranking compared to players locally and globally for a given game can enhance the personalization. This can be done per multiple games and a score can be computed for a player's overall performance across all

the games they have played. An additional feature would be to implement some sort of “1v1 match”, in the sense that in addition to having live/multiplayer tournaments and games, games can be played single player against AI players on a range of difficulties if applicable to that game. These are all features to implement in the future.

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

The final division of labor was equal while we all took different responsibilities of developing the application. Half the group was focused primarily on developing the front end while the other half of the group was focused primarily on developing the back end. For the front end team, tasks encompassed developing the actual Angular components (such as page design, workflows, component architecture, etc.) and ensuring our front end was able to call the backend APIs. The back end team was responsible for writing the advanced database queries, defining the needed APIs and implementing them, and essentially building the Flask application from start while ensuring that it was compatible with our front end. In the end, while these were the divided responsibilities, we helped each other on different tasks and weren’t entirely constricted to these defined responsibilities. Everyone did a little bit of everything and teamwork was managed well with regular meetings and an overall equal share of workload.