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).

There weren't any major changes to our final project based on what we stated in our proposal. We were able to achieve everything we had planned to do for our web application. Due to time constraints, we decided to prioritize and focus on implementing the main features of our web app first which were the budgeting rule, the expenses stats, expenses dashboard, registration/login page and the profile page. As a second version or update, we could enhance the web app by adding more features such as alerts or notifications to enhance the UI and user-experience by notifying users when they go over their budget or recommending them to consider spending less on nonessential expense categories.

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

Our application meets everything that we had set out to achieve when we came up with the idea. We wanted to create an application that could help college students budget their money and track their expenses in a way that catered to their specific demographic and habits. We wanted users to be able to set and update their budgets for the month, input their expenses and track them by category, and create and delete monthly plans that allow them to set goals for how much of their monthly budget they are putting towards needs, wants and savings. We also wanted users to be able to compare themselves to other similar users so that they can get a better idea of how they are doing with managing their finances compared to their peers, and maybe find areas to improve in. During ideation, we made sure to consider what features would be important to users in our target audience. After completing the creation of our application, we can say that our application successfully brought these ideas to life. Due to this, our application will be very useful for college students like ourselves.

Discuss if you changed the schema or source of the data for your application.

In the beginning, we found three datasets from Kaggle that had information about college students and their spending habits based on demographics, habits, and other lifestyle choices. We were planning to use these datasets as the data source for our application, but we ran into some issues because the datasets were not large enough. Due to this, we decided to generate realistic data and created our own csv files to use with the application. The overall schema did not change significantly as our project progressed, however, we did decide to make the Profile and Expenses tables weak entities instead of strong entities.

Discuss what you changed 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?

Compared to the initial design of the ER Diagram and the table implementation, there were some changes that occurred which we viewed as necessary. Initially we did not have any weak entities in the ER diagram, but upon reflecting on the design further, our group believed it was necessary to have Profile and Expenses be weak entities, where Profile is dependent on the User_Account entity and Expenses is dependent on the Profile entity. This is because it would not make sense logically for a Profile to exist without a User_Account and for an Expense to exist without a Profile within our website application. There was another change between the User_Account and Profile entity relationship, where in the final version User_Account and Profile has an one to many relationship, but in the original design it was an one to one relationship. This change was made because in terms of the usage of the website application, we thought it would be better if an individual could create multiple profiles under their one account they registered with. For example, if an individual changed their major or entered a new school year, that user could make an additional profile instead of having to create another account to make a new profile. Not to mention, to make sure there is a reason as to why User_Account and Profile were different entities, having an one to many relationship instead of an one to one relationship was the best decision. With this final version, an individual would have an easier time using the website application. Additionally, in the original design the relationship between Profile and Budget was an exactly one to one relationship and it was the same for Budget and Monthly_Plan (exactly one to one relationship). However in the final version the Budget entity still has a one to one relationship with the Profile entity, but it is now an at most one relationship from Budget to Profile, meaning that Budget is its own entity and not reliant on Profiles to exist. A Profile can choose a Budget, but a Budget will not depend on a Profile. This idea also applies to Monthly_Plans, as Monthly_Plans is not dependent on Budget since there is now an at most one relationship from Monthly_Plan to Budget in the final version. The changes made in the final version compared to the initial design choices for the ER diagram/table implementations is the more suitable design because it addresses any concerns an individual using the website application might have had as well as clears up any confusions on the relationships between the entities/tables that were created for our project.

Discuss what functionalities you added or removed. Why?

Many of the functionalities that we originally planned to include were kept and implemented but there were a few features that were both removed and some that were added. The functionality that we removed was in the monthly Planner web page. This feature would allow the user to visualize their expenses based whether the purchase was a need or want. Now we kept this feature as an advanced query on a separate website but we removed the advanced visual components of the page. The reason we removed this functionality for the user to visualize their expenses is largely due to the time constraint of the project. Our vision for this project was to create a venmo like website that can help students compare their spending habits against other students and track their total expenses. Although the advanced graphs and pie charts that we envisioned would be possible to implement, it was difficult to ensure a high level of quality within the time frame of the project. A user having multiple profiles is a feature we added in the Profile Creation Table after receiving feedback form our TA Lahari. We want the user to have multiple profiles they can look back at to see how much they have changed through the years in school. This was added because it is an additional functionality that brings significant utility to a potential user.

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

Create was used to make a new account, make a new profile, input a monthly budget amount, set a monthly plan, and add expenses to the expense tracker. These were all core functions of our application that were necessary for it to work as some form of basic setup is needed in order to be able to use the application as intended. Read is needed when expenses are displayed to the user on the Home page as well as when other more detailed data is displayed to users in the Leaderboard and Expense Stats pages. This is a basic necessity for the application in order to show the user any data, and many of our web pages would be impossible to use without it. Update was used to change the value inputted for the monthly budget amount, change the goal percentages allotted for needs, savings, and categories in the monthly plan, and edit expenses added in the tracker. This is necessary for the functionality of the application because the amount of money available to spend for the month can change, a user's personal goals can change, and added expenses may contain mistakes that need to be edited. Delete was used to enable users to delete expenses from their history and delete monthly plans that they may have made previously. Expenses need to be able to be deleted for numerous reasons such as if they were added by mistake or received refunds for example. In order for the totals and percentages to be accurate and useful to the user, each change needs to successfully be recorded. Similarly, the delete feature is needed for the monthly plan feature because a user may decide that they do not have a current goal at the moment, and may want to set a new goal in the future instead. Overall, the combination of these CRUD features allow our application to function efficiently and fulfill user needs.

The requirement for constraints was incorporated into our application by having the appropriate primary keys and foreign keys defined for each of our tables. This also meant making sure that the tables we decided to change to weak entities had the correct keys defined. The requirement of using a trigger was met in our application when we created a trigger to ensure that a user could not select a username that was already taken. The use of this trigger was important because username is a primary key for the User_Account table. Additionally, users must input their username in both the Budget and Monthly Plan web pages in order to link the create, update, and delete actions on either page to the correct user. The username is used to find the user's most recent ProfileID as well for these web pages, so it is crucial for each user to have a distinct username. The transaction requirement was incorporated into our application as a separate web page called Leaderboard. This page displayed the results of the advanced SQL queries we made in stage 3, which also meets the requirement of the transaction using at least two advanced queries. The Leaderboard page is useful to users of our application because it allows them to see their Expenses and Budget Overview, which shows the top 15 users who met their monthly plan goals successfully, as well as the Expense Statistics by Major and School Year, which shows more details about the expenses of the top 15 users so that users can get more insight about others who are similar to them. The data shown here can be updated with a load data button. In addition, our transaction also meets the requirement of using a control structure in the form of an if statement which makes sure that the transaction only executes if the condition is met. The requirement for a stored procedure using at least two advanced queries complemented our application in the form of another separate webpage called Expense Stats. The Expense Stats page displays to users information about the Top 'Want' Expenses such as the Amount, Category, and ExpenseID, as well as about the Average Expense by Category per User. The data used in the stored procedure can be reloaded as more expenses are added, and is useful to the user because they can further see how they compare in these categories. Similarly to the Leaderboard page, the stored procedure making the Expense Stats web page possible also meets the requirement of using a control structure in the form of an if statement.

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.

Saavani: One technical challenge we encountered throughout the project was making sure that when one user could have multiple profiles, we were using the correct and most recent one to apply the actions and updates across our pages. We decided to handle this case by assigning a higher ProfileID numerical value to the more recent profile that was created by the user, and then using a max function on each page to ensure that the correct profile was being selected each time. Using this method also made sure that no matter how many profiles were made by a user, the correct one would always be chosen since it would have the highest numerical value.

Jenny: Another technical challenge we encountered while working on our project was navigating the GCP platform, and setting up our environment so that everyone can work on their own page at their own pace. It was a bit challenging having to edit code on the Google cloud shell editor versus using a code editor like VSCode, but it was doable. It also took some time to have everything set up, installing node, ajax, and other backend libraries.

Brian: A technical challenge that our group encountered was understanding how to work with Node.JS and Express.JS for the front-end development. While our group was able to figure out how to work with Node.JS and Express.JS at the end of the project, it was difficult at first since not many of us had experience with front-end development. Additionally, setting up the connection to the mySQL server on GCP in the server.js file was difficult and took some time to make sure it was running properly. Needing to create routes to each webpage within the server.js file was something that caused confusion in our group initially and we also had trouble initially understanding how to make our web pages perform certain actions and how that should be implemented in the server.js file and in the respective ejs file for each web page.

Daniel: One of the technical difficulties we had was inputting the user generated data to the mySQL server. Almost everytime we input the data there was data loss, we would be missing a few rows of data. In this situation we would have to reupload the data into the bucket in GCP before uploading to the server multiple times because the csv may have had empty spaces that were messing with the translation process. Another issue we had was when our teammates assigned the wrong type to an attribute for example varchar for ProfileID instead of Int would cause issues when trying to iterate by one. Luckily we were able to find the issue and change all the tables to make sure it would be an int type instead of Varchar. Creating the data itself was difficult. Lastly I will highly suggest finding already made data on Kaggle instead of creating the data by yourself because it is difficult to account for 1000 users or any types of data while keeping the results realistic.

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

In our application, users can create an account, create multiple profiles, set a budget plan, add expenses, filter expenses, modify expenses, delete expenses, and compare their data to other students and see stats of other student's spending habits. Throughout the different stages of our web app, we did make small changes like updating our ER diagram and schema to allow users to create multiple profiles, and making a few updates of our database tables to allow auto_incrementation of ExpenseID's, but overall, our final application encompasses what we had envisioned it to be in our proposal.

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

In the future we would like to add a direct message feature that allows users to talk with each about their expense habits. I think this would be great because it would have more functionality than Venmo has. In accordance with Venmo we would like to add a feature that allows the user to comment on their friends expenses so that people can laugh about "funny" expenses. The final functionality would be the friends functionality that would allow you to directly compare expenses with their friends. Some of the other things we would want to refine are the limits on the advanced queries because we believe the amount of records displayed to the user is too much.

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

The final division of labor was evenly distributed for each step of the project and our team was able to manage our teamwork well at the end of the project.

For coming up with the project concept and working on creating the web pages for the project, everyone was able to contribute evenly. Every member contributed to answering the questions that were asked during Stage 1. Jennifer and Brian created the initial design for how the website should look and what should be displayed on the website using Figma to create the design.

Creating the ER diagram and explaining the different parts of the ER Diagram for Stage 2 were done by every group member. Because the data source we chose ended up not having enough records for our project, Daniel and Brian ended up creating the auto-generated data in the format of CSV files to use in the project.

In regards to Stage 3 of the project, setting up the GCP VM Instance and the mySQL server was done by Brian. Writing the DDL command and creating the tables based on the ER diagram in the mySQL server was done by every group member. Uploading the auto-generated data to the mySQL server was done by Brian and Daniel. Uploading the web pages required in the project to the VM Instance was done by every group member. Every group member was also responsible for creating an advanced query on their own, explaining what the advanced query does. and running it on the GCP mySQL server to get the top 15 results. Creating various indexes on the mySQL server for each advanced query was also done by everyone, where each group member was responsible for creating indexes based off the advanced query that individual created, getting the performance data before and after creating indexes for the advanced query using EXPLAIN ANALYZE, comparing the performance of different indexes, and then choosing a final index design for the individual's advanced query.

Lastly for Stage 4, each group member worked on the front-end and back-end parts to make sure the website application works properly. Jennifer was responsible for the front-end and back-end code for the Expenses page, making sure there was a feature for updating expenses, deleting expenses, and searching for specific expenses using a filter using the search bar. Saavani was responsible for the front-end and back-end code for the MonthlyPlanner page, making sure the option to add and delete monthly plans was working. Brian was responsible for the Login and Registration pages and their respective front-end and back-end code, ensuring that registering with a unique username and logging in with the correct credentials was working properly. Daniel was responsible for the front-end and back-end code for the Profile page, making sure that a user can create multiple profiles and that this is reflected correctly in the ProfileCreation table in the mySQL server. Daniel, Brian, and Saavani worked together to create the Budget page, making sure that adding a budget from this page worked properly and any changes were reflected in the Budget table in the mySQL server. Brian worked on the Leaderboard page and the front-end and back-end code for this page. The Leaderboard page required using a transaction, so Brian ensured that the transaction had valid control structures, correct isolation level, and included at least two advanced queries. Daniel created the front-end and back-end code for the Expenses Stats page. The Expenses Stats page required using a stored procedure, so as a result Daniel made sure that the stored procedure had at least two advanced queries and correct control structures.

Overall, our team managed teamwork well since every member contributed at each stage of the project. We also had frequent meetings throughout the semester to discuss any questions on the project and what tasks each member should take to contribute to this project, which everyone showed up for and contributed.