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).

Summary: Our original project proposal was focused on visualizing flight delays for US airports based on airline or route, but we weren't super clear on how we would measure delay. Our final project focused on the average arrival delay as the key metric. It also added additional information like average flight price.
Creative: Our original project proposal wanted the map to be interactive based on the filters that users applied based on delay time and origin/destination. The map should be able to display cities and routes and have both of those objects clickable to display more information. Our final project improved upon this with color visualization.
Usefulness: Our final project does not differ much from the original– we ended up having both an interactive map to filter flights as well as a Favorites list that users could add to.
Data: Our final project used more datasets than originally intended. The original plan called for just the Flights and Cities datasets, but the final project added the Price, Airlines, and Airport Coordinates datasets.
Functionality: Our final project had more functionality compared to the original plan, as we also included filters and CRUD for price information as well as delay information.
UI: Our final project had a similar UI to the original proposal.

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

Our application **succeeded in integrating advanced queries** and database methods as well as **visualizing different flight metrics in a succinct and intuitive manner for researchers** and other professionals. The visualizations by city and by route led to many insights when it came to patterns for those two groups and corroborated past observations about airlines. For example, Frontier has most of its flights in Denver, which happens to be where it is most popular. Of the airlines sampled in our dataset, most of them had a similar proportion of flights that arrived on time, meaning that most of the issues that people have with some airlines come not from the average delay but from the service and other amenities. These are useful findings in aggregate and are easily revealed to users through use of our application.
Our application had **room for improvement when it came to direct user relevance for day-to-day consumers** since all the data being displayed were aggregated and not updated in a timely manner. This means that typical users looking to use similar applications to book flights, would rather just use sites like Expedia to look for flights that get them from point A to point B and find the cheapest versions of those. For most users, conditioning on the cheapest/least delayed flights first has less utility.

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

Entities: Our original proposal had separate entities for Flights and Delay and did not have a Routes entity. In our final project, we combined flights and delays into a single table, and added a Routes table which encoded the different origin and destination airports that a flight could take. This greatly improved query efficiency involving delay, since we did not have to join 2 tables with 1 million+ rows.
Routes: We also made the decision to have the Routes table include the *averages* for price and delay. This way, to display the routes in the Routes Map, we only had to query Routes (less than 2000 rows) instead

of Flights (over 1 million rows). Whenever a flight was added to Flights, we dynamically updated the Routes table to reflect the newly updated averages.

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?

Our original ER diagram was very different from the final version, mainly because we needed to add new relations and refactor them based on new datasets we wanted to add. The original diagram made the assumption that a flight number could uniquely identify a flight when in reality it could not, so we used origin, destination, and time as other fields to comprise the primary key in the final project. The original diagram also had Delays as a weak relation, but we ended up merging delay and price information into Flights since it was easier to look up. We added a Price and Airlines dataset and thus needed an airline relation in the final project.
Ultimately the final project's relations ended up being a more suitable design since they were more convenient to query when it came to frequent requests such as filtering by price or delay time with routes. This way, the queries that users would be more likely to use happened much faster while the advanced queries, which were relegated to their own page, could be optimized to be faster.

Discuss what functionalities you added or removed. Why?

The main pieces of functionality that were added were the route coloring on the map, the flight metrics page, and general map interactivity. We also did not have to remove any of our initially proposed ideas since they were all required for the base requirements of the project and were all eventually implemented.

We added route coloring to make the map easier to read and show the difference between displayed routes based on the price and delay after setting the filters. The flight metrics page were actually required for the more advanced queries that were required for the project as our initial proposal did not take all of the requirements into account. The data displayed for airport and airline metrics included the stored procedures and transaction queries that were necessary. Additionally, we made some edits to our proposed UI in order to include more user interactivity by adding the ability to click on routes that are displayed and the circles around the airports to get more information about the flights on the routes and the airport statistics.

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

Trigger: These allowed us to update the average metrics for price and delay of flights, which was most readily seen in the Favorites section. This allowed for integrated interactivity and updating of data so that it could be most relevant for the user.
Transactions: These allowed us to group multiple queries that had to run together, which was particularly relevant for advanced queries. This was particularly useful for our airline timeliness metric, which displayed airports and their proportion of early/late flights.

Stored Procedure: Our stored procedure was necessary for the airport traffic metric, where we had to get the count of incoming and outgoing flights for each airport and then find the average delay for each airport as a destination and as an origin airport. Stored procedures allowed us to more succinctly define our advanced queries into one function.
Constraints: PK/FK definitions and indexing were crucial to our application since they greatly affected the speed of our advanced queries. Some queries saw a reduction from dozens of minutes to just a few seconds of runtime due to indexing, and constraining our database allowed us to display those valuable, interesting insights to users.

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.

- Tyler: The whole process of having the client make requests to the server, and having the server send data back to the client, was new to me. This was my first real exposure to full-stack development. I got very comfortable with making requests from the front end and handling those requests via view functions on the server. The server would make SQL queries based on form data that was sent in the request, and I would send the final response back to the client in json format. Before implementing everything all at once, I found it very helpful to incrementally develop different portions of the project. For example, instead of trying to perform a SQL query and send the results back from the get-go, I would first ensure that the frontend request was directed to the correct route handler and that the sending of data was working as intended. Only then would I start working on our queries and implementing real functionality.
- William: Both processes of working with frontend layouts and handling SQL queries & optimization were new to me before this class. The largest technical difficulty was trying to optimize the advanced queries to run in a few seconds instead of taking minutes. This took a lot of trial and error at all stages of the pipeline, from recleaning and editing the data itself to trying different relations as well as indexing combinations. I found it helpful to document the exact combinations of parameters and data that had been done before as well as drawing out the schema and really getting to know tangible rows of the data to be helpful.
- Kevin: One of the biggest challenges for me personally was applying the more theoretical SQL concepts I learned through classes, lectures, and homework to a real-world project. Because of how my previous practice with SQL had been in a controlled setting with well defined table schemas and logical ordering of information, there was a little bit of a learning curve for me to come up with schema that made sense of the information I was presented with in the data sources. Moreover, I had to consider things like optimizations (we don't want tons of joins when we call some of our most popular queries) and the best way to ensure that our schema was flexible enough for cross-table queries if the need arises. One aspect of working with real SQL databases was also the idea of future-proofing and thinking about how we could potentially modify our schema to accommodate changes or additions to our data sources.
- Abhitya: The biggest challenges for me in this project were the optimization of SQL queries and the work required to make a functioning front end. Most of my work with data has been through more OOP languages and this was my real first exposure outside of self-study to working with

query languages. Even though the class gave a good introduction to working with database systems, I think the implementation of this project was a lot more difficult than I initially thought. I think the main piece of advice I would give regarding this project is to truly understand the data you have and try to properly connect it with the final project design you wish to have. The main technical challenge for me was figuring out how to do the front end for this project, I had never worked with that aspect of a project before and had to learn my way around it and work with my teammates to get a better understanding of how to set everything up and how to even complete the tasks at hand.

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

Aside from what was discussed in the added functionality section the general styling and page layout were changed from our initial design but nothing else was really modified.

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

Future directions to take this application could extend to other modes of transportation such as maritime vehicles or automobiles and highways. Other data the application could take into account could be weather, passenger sentiment by airline, etc. To shift the usefulness of the application from researchers and those looking at aggregate data to typical consumers looking for the latest data, the application could incorporate live price metrics and other measures of demand.
Future database optimizations could be refactoring relations and chunking up queries even more to see if there are ways to get our advanced queries to run even faster.

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

- Tyler: I was in charge of implementing the event listeners on the client-side, and ensuring that requests were properly handled on the server. I would use the SQL queries that we came up with in previous stages to aggregate data that would then be sent back to the client to be visually displayed via map or chart. I was lucky to have found libraries that were very useful in visually displaying our data. I used the Google Maps API, as well as Chart.JS to enhance the quality of our project's main features and make the development process easier. Regarding the event listeners and view functions, pretty much every function of the web app relied on some part of the client-server architecture. Whenever a filter or form was submitted, or for anything that required a SQL query, it required setting up a route handler on the server that would interact with our SQL instance on GCP.
- Kevin: I was mainly responsible for a lot of the frontend work which started up with setting up template for a Vite (JavaScript compiler) + React.JS project. I chose to use Vite as our local development server instead of something more popular like Create React App because it introduces HMR (hot module reloading) for easier frontend development and that would prove to be invaluable because of how dependent our project was on an intuitive frontend that would serve to display the data we were interested in. I also did a lot of system design work to figure out exactly how to fetch data from our SQL database (the frontend + backend server architecture that

we ended up going with) and choosing a lot of the technologies we would be working with. As with the rest of the group, we all worked on the initial design and schema, finding data sources, and ideating on a project of choice. There were no major struggles in term of teamwork and we were able to meet in-person regularly to exchange ideas and help flesh out a plan with feedback from all the members.

- Abhitya: As I mentioned in the biggest challenge, most of my work was done on the front end of the project. I also contributed throughout the semester to the theory and ideas behind the implementation of the project, coming up with potential ideas for the project based on the dataset we found. When it came to the final design of the project I mostly worked with Kevin to make a robust front end that provided the users with a good experience and was easy to understand and move around in, helping a bit with the actual code required for connecting to the SQL instances and mostly being in charge of the overall final appearance of the site developed.
- William: I was in charge of a significant portion of the frontend structure DOM layout, as well as handling query optimization whenever we had to refactor our relations to add new data. Tyler and I would coordinate how we wanted to change our Stage 2 SQL queries to add in new datasets or refactor relations for efficiency. For the advanced queries, I was responsible for indexing and otherwise optimizing them to run faster.