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).**

   We had originally thought of users being interested in trends regarding best times to fly, but often, as we came to understand, delay/cancellation-sensitive travel tends to be urgent, meaning users would be more interested in the most efficient way to get their destination right now rather than when to postpone their trip to in order to avoid delays and cancellations. — *(Same as our Final Demo answer)*

   We also had decided on including a map feature (which would have been our creative component) but chose not to do so due to time constraints and relevancy to the application.

   We expected to have a table view incorporated for rankings but instead took user input on the number of results they desired and returns those in list format.

   We thought that our search bar would search based on airports, airlines, and routes to uncover rankings and related data, though our implementation focused on searching based on airports.

   Our UI mockup differed from our actual interface due to the removal of the map (creative component), the drop-down menus for search (we searched based on IATA code instead), and by separating all of this functionality into its own separate pages to consolidate related items.

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

   We believe that our application was successful in creating an effective ranking tool for airports based on historical data, including aggregating attributes of this data to uncover other meaningful rankings. Thus, a user can easily figure out which airports are notorious within the US for untimely flights and avoid them. We also believe that our application, unlike FlightRadar24's service, incorporates the data of a far greater number of domestic airports, allowing these rankings to be more accurate for the relevant airports than the competitor. We failed to provide data by route to help users narrow down an airline, so our application is only useful to those unsure about which airports to fly between. Moreover, our application was meant to use this historical to uncover weekly and seasonal trends to recommend to users, which we did not have the opportunity to incorporate due to time constraints.

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

We did not change source of data for our application. We changed our schema in two significant ways when performing our table implementations. Firstly, we removed foreign keys, and secondly, we combined the delays and cancellations table to instead be one table reflecting untimely flights.

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?**

   We changed our ER diagram in two significant ways when performing our table implementations. Firstly, we removed foreign keys as we realized that they were not necessary as identifying attributes our table rows. Instead, by simply joining on those attributes using natural or theta join, the same data could be read and manipulated. Additionally, we combined the delays and cancellations table to instead be one table reflecting untimely flights. This is because most of the trends that we planned to uncover incorporated both attributes when gauging the timeliness of an airport, so splitting them would not have been useful. Our updated design is certainly far more suitable because we require fewer SQL commands in our queries to provide the same functionality to our users.

5. **Discuss what functionalities you added or removed. Why?**

   A functionality that we removed was the map display routes (the creative component). This was primarily due to time constraints, but one significant direct contributor to this was the fact that we designed our application heavily around choosing airports with the least delays as opposed to routes. Therefore, a map of airports would have just been a generic map whereas a map with routes overlayed would have been more informational. Therefore, as we pivoted from calculating statistics on these routes to calculating them on airports, the map ceased to be as relevant and useful. We also planned to have an optimization algorithm (one that would find a route that minimizes delays), but this was also phased out as we pivoted to airports. Separating our application into windows with relevant functions was a functionality that we added. This was primarily due to the changes we made to the original UI Mockup, but it also served to improve navigation for our users.

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

Our advanced database programs complement our application well in their individual ways. Our stored procedure provides powerful insight into how delays occur at airports by checking the arrival delays for flight at an airport that arrived late, uncovering the ripple effect of delays to rank airports based on true vulnerability to delays, which none of our competitors are currently capable of. Our trigger ensures the integrity of the aggregated values and statistics that we provide by nullifying any outliers before they are added by users to the database.

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.**

   Yuan - One technical challenge could be ensuring the security of the application and the database. When building a user interface that interacts with a database, it's important to implement security measures such as a middleman between the interface and the database to prevent unauthorized access, data breaches, and other security threats. We originally thought of putting the SQL queries in our HTML, but, as our TA pointed out, this would have been dangerous. Thus, by creating a Node.js server, we avoided exposing our database implementation to the Internet.

   Alex - A technical challenge we encountered is scalability. Database servers, including ours, have resource and storage limitations. If we wanted to include even more flight data in our dataset than just 2015, we would run into scalability issues as queries would repeatedly have to aggregate so much data. To resolve this, we would work on optimizing SQL queries and implementing indexing strategies. For instance, we could build upon the B+ tree and Hash Table indexing methods learned in class. Thus, we can reduce the volume of queries that we are running.

   George - One technical challenge that we encountered when creating our website was the integration of frontend and backend. The original frontend that the team created was insufficiently prepared for the integration process, so significant reworking was required to ensure that the whole thing worked smoothly. Coordinating better to design both parts to work together seamlessly from the beginning would have saved time and effort.

   Aryan – A technical challenge that we encountered was when trying to implement and index the database on GCP. The cloud shell was very confusing and uploading values to the tables required a very specific formatting that we were struggling to figure out. Fortunately, through the mySQL command line interface, we were able to set it up properly, and once we understood how to use the mySQL command line interface, we

figured out setting up the Cloud Shell from the documentation and were able to run everything from within the GCP tab itself.

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

   No other changes were made between the original proposal and final application that were not already discussed in the questions above.

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

   (Same as our Final Demo answer)

   Since we find so many aggregate values (averages, sums, etc.) corresponding to airports, we could add that as a column to the airports and just query that static value instead of aggregating to find the same value every time a user wants it computed. We could create a stored procedure to update that static average or sum whenever an insert/update/delete command occurs on the flights table.

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

    The final division of labor was generally consistent with our original proposal. Up to and including Stage 3, we met up in-person or on discord and worked together on the assigned tasks. For constructing the interface, we split ourselves up into two groups. One group, Alex and George, worked on writing the HTML and CSS of the interface. Another group, Aryan and Yuan, worked on building the server in Node.js and integrating it into the frontend. Overall, we managed our teamwork excellently.