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).

One of the changes we made from our original proposal was removing the dynamic A-F grading system with user-ranked features and visualizations. The final version now uses a 1-3 ranking system with only two preferences: temperature and precipitation. Users have to input their preferred temperature value and select precipitation preference between low/medium/high rather than ranking multiple features by importance.

Also we changed our plan of having a range of destinations to compare to making it strictly just 3 locations to compare.

Reduced Location Comparison: Instead of comparing 2-4 destinations as initially proposed, the final application strictly compares exactly 3 locations. This constraint simplified the ranking algorithm and UI design.

We also dropped our optional feature in which we would try to integrate OpenAI API integration into our program as the complexity and learning of all of the documentation would have been difficult in our given timeline.

Finally, we added a 5-trip storage limit which we originally hadn't planned in order to have a space limit on our database when we create trips.

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

Achieved:

- Created a functional tool that addresses the core problem of users compare destinations using historical weather data rather than unreliable forecasts
- Successfully integrated real-world datasets into a queryable database
- Implemented a trip saving/loading functionality that allows users to revisit their past creations
- Provided date-range specific comparisons, making recommendations relevant to actual travel periods
- Added specialized queries such as locations with no bad days and above-average precipitation locations that offer unique insights not available in standard weather apps

Shortcomings:

- The updated preference system doesn't capture the full complexity of travel decisions we had originally intended such as humidity, wind, air quality
- The ranking algorithm lacks transparency—users see final rankings but not the underlying data or scoring breakdown

- The application doesn't explain why a location ranks higher, missing an educational opportunity

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

The database that we were accessing was extremely large as it was originally storing the data of every weatherstation in the US so in order to do that we parsed the data to only include stations that had temperature and precipitation reporting guaranteed. We also limited the scope of our stations to be in the mainland US + Alaska/Hawaii, so as to simplify our frontend to only include states. The schema was modified to allow for more efficient queries. We made the decision to incorporate new RND_LAT and RND_LNG columns to allow for easier spatial comparisons by grouping nearby coordinates for range-based queries. That way we don't have to calculate if each LAT/LNG was within a radial distance on each check of each row.

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?

The transition from the initial ER diagram to the final DDL implementation involved several changes/refinements to optimize the efficiency of our queries. A significant change was the removal of the locationID primary key in the LOCATIONS table, replacing it with a composite primary key of CITY and STATE to enforce uniqueness and simplify joins. The HISTORICAL_WEATHER table was also restructured to key directly off LAT/LNG and DATE rather than a separate ID. Furthermore, the AIR_QUALITY table was removed entirely to reduce complexity, while the PREFERENCES table was streamlined to focus on preferredValue rather than just rank. These modifications meant we could create a more performance-oriented schema, particularly suited for the efficient geospatial lookups required by our project, as that was a huge performance hit.

Discuss what functionalities you added or removed. Why?

Removed:

- Removed the feature where users could rank 4 features by importance
- Did not create visual representations such as graphs for locations.
- Removed the letter grading A-F scale to rate locations for users.
- Removed the idea of an OpenAI API integration
- Removed the flexible location count to have a preset value of just 3 locations

Added:

- Our original proposal had saving trips but didn't specify the 5-trip limit with automatic oldest-trip replacement
- Added the trip modification feature to saved trips which was not clearly defined before

- Explicit Date Validation ensuring that start date must precede end date, enforced in the UI
- Added a city existence feature that prevents users from querying non-existent locations

Why:

Removals: A main reason we removed some of the features was because of the extremely large dataset being very hard to repeatedly search for some of the features we had planned. Additionally, we were also a bit time restricted and building the ranking algorithm, visualizations, and air quality integration would have required significantly more development time

Additions: We ensured an automatic oldest-trip replacement so that there would be a seamless transition if the user wanted to create more than 5 trips which is not allowed. We also added the trip modification feature to display more complex SQL usage so that tables can be edited after they were created and items that were not explicitly changed also adapt to other constraints. The trip management features emerged from implementing CRUD operations and discovering UX needs (like preventing invalid cities)

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

The core ranking and comparison features depend on multi-table joins and aggregations over historical weather data to generate the trip rankings. The “no bad days” and “no rainy days” queries add further insights that go beyond standard weather tools. Database-level constraints such as the 5-trip limit, along with indexing decisions, support data integrity and ensure the system remains performant as usage grows.

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.

Aditya: Cleaning up the data to be in a format that was representative of our application took a large amount of effort. We had underestimated how long it would take to make the scripts that modified data to be in the right units and removed null/incomplete values. I would suggest exploring the data and the datasets as soon as possible, such that you can either preprocess the data as is (with best practices), or change the scope of your project before it is too late.

Ali: Creating the right SQL commands and integrating them into our backend required a lot of trial and error. Specifically, developing queries to access the right locations were continuously failing due to calling the wrong coordinate ranges. These calls were also very slow since it needed to search across a range, rather than look for certain values. We overcame this by making a new column in the database with rounded coordinates, then rounding a city’s coordinates and looking for direct matches when searching for them.

Michael: Initially, we recalculated trip rankings every time a user loaded a saved trip, which added a significant amount of compute time and overhead. We resolved this by storing the finalized rank order in the database at save time, so loading a trip was a direct read from the database rather than a recalculation. I recommend for future teams to store computed results in the database when possible.

Ojas: When creating the frontend for the application, we ran into a problem with addresses as we forgot to integrate the base API url into our frontend pipeline resulting in a quite bit of debugging until we found the issue and changed all of the addresses to the correct one which was just the one we had before with the API url in the prefix of the address.

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

We originally didn't emphasize how much detail we would put into the authentication mechanism. For our final implementation we use a simple username/password system checked via POST /check_user_login rather than more secure approaches however if we were to deploy this on a larger scale we would put more emphasis on security.

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

Future work beyond the interface could focus on expanding the analytical depth and robustness of the system. We could incorporate additional decision factors (e.g., humidity, wind, air quality) and extend the schema to support richer, user-weighted preference models. The ranking algorithm could be made more transparent and evaluable by storing intermediate scoring components and adding explainability outputs. On the data side, integrating forecast data alongside historical averages could enable hybrid recommendations, while improved cleaning and missing-data handling would increase reliability. Finally, performance and scalability could be strengthened with geospatial indexing, partitioning large weather tables, and caching frequent date-range queries.

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

Aditya and Ali managed the backend and logic. Michael and Ojas worked on the front-end UI design and development. Everyone was involved in the SQL ideation. We managed teamwork by having planning meetings, sync-ups to address, determine the direction of our project, and by prioritizing communication by updating one another when we were complete with our portions.