- 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).
- Discuss what you think your application achieved or failed to achieve regarding its usefulness.
- Discuss if you changed the schema or source of the data for your application
- 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?
- Discuss what functionalities you added or removed. Why?
- Explain how you think your advanced database programs complement your application.
- 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.
- 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
- Describe the final division of labor and how well you managed teamwork.

1. **what we have achieved and failed**

In our original proposal , the description of our project is a platform allowing user to access and display multiple bookmarkable locations to see the weather information, and receive recommended outfits consisting of the items in their virtual wardrobes based on current weather.Users should be able to subscribe multiple cities to their favorite city lists and add items to their virtual wardrobes. Overall ,we achieved all the expected functionalities . Our web application allows users to log in , access to their favorite city lists , subscribe to different cities by searching keyname,and generating recommended outfits based on the weather of cities they subscribe to, as well as adding , editing and deleting items to their accounts. Our web application also provides data visualization , including line charts of the average temperature over days and a map showing all the cities in our database with a tooltip showing the city's information.

2. **what change we made to schema,data source , ER diagram , or table implementations, or functionalities.**

For data sources, we had considered real time weather API for getting data to insert into our tables. However, this posed a challenge in that we would have to periodically run an automated task to update our tables in real time. In the end, we settled on an extensive weather dataset from an older date since it had more data for more cities in the USA. This would make piloting and testing our functionality easier. Now, plugging in the real time API data through another script in python is trivial since the rest of the application is working as expected. We made this compromise in order to deliver on the deliverables expected from us for this stage.
Changes: For the ER diagram, we made the following changes over the course of stage 2: weather is uniquely identified by the location of the weather. For eg. "Snowy" isn't unique since multiple locations can have the same snowy weather. So "Snowy","Urbana" is a unique combination. This wasn't the case first. Made user-list into a many to many relationship between user and weather, thus the schema remains the same but the relationship is more clearly defined.Added more information to Location entity set.Now State is uniquely defined by City_Id and weather is uniquely defined by City_id and date. For the final application, we had a separate table for outfit and the relationship outfit_has_apparel in stage 2. We

combined that relationship into the outfit table in the db in sql since it makes it easier to do queries on that table. Otherwise we would always do a join with outfit and outfit_has, only increasing complexity unnecessarily.
Functionality wise we ended up covering most of what we intended, which is something we are proud of.

3. **Explain how advanced database programs complement applications.**
   Advanced database features like stored procedures, triggers, and transactions furthered our application's performance by streamlining data manipulation and ensuring integrity. For example, we used triggers in our weather wardrobe project to automatically update or log changes without user intervention. We used stored procedures to speed up data retrieval by handling complex logic directly in the database, thus reducing network load and enhancing performance. We also used transactions to protect data integrity by ensuring that multi-step operations either complete entirely or not at all, which was needed for maintaining accurate database states during processes like user registration.

4. **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 were to maintain your project.**

   Qiaochu : one technical challenge I encountered was passing the user 's id from login page to user' s page.After logging in , a user should see their own pages and their subscribe lists thus our web application should read and return the page corresponding to current users' id. We solved this by using session and saving the user's id from session to a variable when log in ends , then passing it to other functions.

   Hammad - One technical challenge that I encountered was with the implementation of the graphs that are displayed for each user for their specific subscribed cities. I was struggling with the application detecting which user was logged in which was required to display the correct graphs based on the user's specific "usersubscribeslist" which highlighted their bookmarked cities. To fix this issue, I used a middleware method so that the user 's id would be saved after logging in and can be accessed by the other pages of the web application, so I would be able to access the user's id in the graphs page and display the graphs for a specific user's list of cities.

   Ibrahim- A technical problem I faced was correctly updating the users subscribed city list to the correct user. More specifically, I had difficulty managing user sessions and ensuring required data like 'userID' was consistently being passed between the server and client. The error I was facing was a "userID not defined" which stemmed from my misunderstandings in how server-side variables are accessed in client-side JavaScript.
   Venkat- A technical challenge us as a team encountered was connecting to a MySQL instance in gcp from local node. Initially we made a connection through a VM in GCP which worked well. However, we ran into an issue where we ran out of money just before CP1 demo unfortunately. So in order to reduce costs, we decided to not use a vm on GCP and use our local systems to run the node application. In order to achieve this we needed to get a gcp proxy running, something I set up first and helped other teammates do so as well. All of us had different local environments, so getting the setup for the gcp sql connection working took some work. Also, another challenge was collaborating effectively with github. Setting up the repository was easy, however merging all the different versions of code with merge conflicts

took some troubleshooting. VS Code Merge editor helps, something like Git GUI would help people new to collaboration with Git manage this effectively.

5. **Describe future work that you think, other than the interface, that the application can improve on**
   Future work can focus on improving the user's favorite city list. Currently users can't delete cities from their lists but they can only add and view their favorite city lists. So far, we have built a subscribe log keeping track of how many times each city has been subscribed and when do users subscribe to them. In future, we can make it accessible to users and recommend users cities that they might be interested in based on their data. Adding real time weather data API to the table periodically using an automated script would make the weather data set up to date. Adding image functionality where we can store images of clothing items in a database might make the application more intuitive. Having more weather metric to make a more informed decision might help. Also using an ML algorithm to give recommendations based on training data of weather and outfits, which can self improve using data generated by the application could be a cool technical addition.

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

final division
Venkataramanan: Worked on the page that did the CRUD operations on Apparel data. This includes the SQL queries, front end page to fetch, update, create and delete entries that reflect in real time in the front end page. Created the stored procedure to generate outfits and the trigger that triggers the stored procedure when a new city is added to the user's list.
Hammad: Worked on the front end and back end design for the graphs, and weather pages. Also, I created some of the advanced queries and implemented the transaction method for registering a new account. I also implemented the API that would display the interactive map that would show all of the cities that can be accessed for our application.
Ibrahim:Worked on the frontend and backend for the 'search for cities' page. Developed the MySql queries and api endpoints to fetch the information (the city names, their respective cityIDs, and the cities that the current logged in user is subscribed to) from the database and display it on our frontend. Also developed the api endpoints to update our database whenever a user added a new city to his subscribed list of cities.
Qiaochu: worked on front end and back end design for the login and register page , designed the trigger which keeps track of the subscribed cities to generate a subscribe log and calculate the most popular cities

We managed teamwork by allocating different responsibilities to each other through a discord group chat. We scheduled times to meet together or with TAs to resolve possible or concurrent conflicts. Also, we communicated our progress through our discord group chat. We tried to do "agile" by assigning "tickets" to each other as different front end pages, different queries to work on the indexing, different sections of a document and follow up on it when we have periodical calls. Whenever we had blockers we would discuss them and get them fixed with help from the others. We setup Git repository with the latest code and had separate feature branches for each of us. Once it was done we merged them periodically with master and kept a clean main branch that was the working latest code.