Team 006: DataBased
Shon Shtern
Angela Roeser
Nathan Grace

Please list out changes in directions of your project if the final project is different from your original proposal (based on your stage 1 proposal submission).

While we did manage to implement many of the primary features described in our initial proposal there were also several revisions made in the process of designing and deploying this application.

The most prominent change is that we implemented a simpler version of our initial idea to allow the user to input stop and end locations, and generate a route. While this is certainly possible given the data, it would likely have required a much more intensive investment of time in order to find all available paths between to stops, and then recommend a route based on some kind of shortest path algorithm like Djikstra's or similar. This might be made more feasible if we were using a node based system such as Neo4j, since that structure is graph based and lends much more naturally to "pathing" queries.

We were able to include user accounts, and the ability to update or delete user details. However we did not manage to implement any complex manipulation of the saved trips the user designates. Given more time we would almost certainly want to implement a "remove trip" function at the very least. Also, while the users do have accounts we were not able to implement a true user login function like we initially hoped due to time and complexity constraints. Doing this would also be a priority for future improvements as it greatly streamlines the user experience.

Estimates of fare cost or total trip duration were not included because without a full transit path it is not certain what the true results would be. We also could not recommend the "closest" stop for the user as we have no way to approximate their actual location without obtaining actual geolocation data for users. However we opted to include a "frequency" measure to inform the user how long they could expect to wait between vehicles on any given trip, allowing them to choose the most time efficient option.

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

I believe our application succeeded in a lot of ways. The data set features public transportation in Sao Paulo, Brazil so it is helpful for users to be able to plan routes that utilize these features. Public transportation is a public good because it allows for people to reduce their carbon footprint while still being dynamic in their city, reduces the amount of congestion in the downtown area, and empowers people to explore the city around them.

By sharing information with the user interface, we allow the users to access the data in a protected way such as our route lookup and our stop lookup. In addition, we have a keyword search to help the users be able to locate stops even if they don't know the full name to assist in their travel planning. We have also granted them access to saver trips in their own personal portion of the database in the Travels table.

The creative feature of the badges adds a community element to our application. It encourages friendly competition for people to plan trips and also keeps users engaged by keeping up with the quartile ranks of gold, silver, bronze, and basic and encourages diversity and routes with the rainbow rider badge.

Regarding the back end, our data is well organized and the tables are linked together with foreign keys and so we were able to extract data in an efficient manner. We have added indexes to speed up the querying process; a lot of our queries we have to combine four or five or more tables so that we are able to do the joining quicker for the benefit of the user. We have made sure that our entities are connected in a thoughtful way so that there are no redundant dependencies.

An area that we could definitely improve upon had we more time for this project would be the user interface. There are a lot of instances where our user has to continuously enter their email and or their user ID, so if we had more time we would be able to implement a login session so the user would not have to continuously enter this information. This would be a more streamlined experience for the user.

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?

We made a few changes to our schema and diagram throughout the project, all of which apply to the ER diagram. First, we updated the date attribute Travel table to a DATETIME type. This ensures we are able to obtain the dates and times in which the person wishes to ride the bus. Additionally, we updated trip_id to VARCHAR 30 instead of CHAR 9. This is because we discovered some trip ids that were of size 10, rather than 9. Thus, we updated it to VARCHAR 30 in the case where we insert larger ids, and 30 characters seemed sufficient for this requirement.

Next, we changed the primary key in Shapes to be shape_id and shape_sequence, forming a composite key. This allowed Shapes to have a weak entity relationship with Trips, which makes more sense given the description of the relation. This is seen in the many-one relationship between the two relations. Similarly, we changed the Stop_Times primary key to be trip_id and stop_sequence, creating a weak entity relationship with Trips, which can also be seen in the diagram.

We also created a new Reward table. The Reward table had the following attributes: row_id (which acted as a unique primary key), user_id (which held the id of the user), reward_type (which held the type of badges each person had), and reward_value (which held the level of their reward, like gold/silver/basic). This was implemented as a many-many relationship with Riders in our database. However, in the future, and as seen in the ER diagram, it should be converted to a weak-entity, many-one relationship with Riders in terms of achievement, with the foreign key being the user_id. This is because each rider can have many badges, but badges can only be tied to a specific rider based on their attributes.

Discuss what functionalities you added or removed. Why?

The functionalities on our user interface include allowing a user to sign up, change their email address, delete themselves from the database, look up stops with a keyword search, look up routes with a stop ID and a date time and with an integer representing the bus, save that trip in the travel table, and see all their previously saved trips. We felt that this was an appropriate amount of autonomy that the users could interact with the database in a safe way. The users also have the opportunity to earn badges such as a rank badge and a rainbow rider badge. Through some Advanced queries, stored procedures, transactions, and triggers we are on the back end are able to do some aggregations in order to split the users into categories and return relevant badges. We thought this would be an engaging way to drum up some excitement, as well as some friendly competition.

A functionality that we ended up removing in between stage one and the final stage was having users enter coordinates and discover a stop closest to them. Due to the time constraint and wanting to make sure that we were able to complete the stages in an effective and efficient way we opted to remove this in favor of a keyword search. In addition to the keyword search being a part of the rubric requirements, the keyword search also better fit into our skill sets. We felt very comfortable implementing the wild card search term in the query that would return anything that was similar to the user. We believe this to be an appropriate substitution because it could be the case that someone knows part of a stop name but not the entire name.

Another functionality that we ended up amending was we wanted users to be able to see the busiest roads and the busiest routes. Our substitution included the average interval between frequencies on steps in our lookup routes stored procedure. This way if someone is wanting to use a stop at a certain time of day they're able to see the average amount of time in between frequencies so if they miss one route they're able to see that on average another one will appear within a certain amount of minutes. We felt that this would provide more accurate and detailed information than simply which roads are the busiest.

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

A major premise of our application is creating a useful and fun way to engage with the Sao Paulo transit system. To achieve this we developed the concept of user badges, or rewards, which can be earned by completing certain sets of activity within the application. Riders are ranked by quartile into Gold, Silver, Bronze, and Basic categories, based on their usage of the application's trip saving feature. They can also earn a “rainbow” badge for saving 7 or more distinctly colored routes, and a King of The Hill/ 1 in a Brazilian reward for acquiring both Gold status AND the rainbow badge. This enhances the user experience by offering a fun, scavenger hunt style experience to what would normally be a mundane task, while also offering a material benefit by allowing the user to save trips for future reference without the need to search for them again.

In order to facilitate this system of rewards, we had to develop a database program that could accomplish all the necessary updates for user rewards as they add more trips to their saved list. Because the user rankings can change with each update, we accomplish much of this via stored procedure using an isolation level of Repeatable Read. This procedure is used to add records to the Travel table when a user saves them, and then calculates potential awards for all users, since any insert can potentially reorder the entire ranking. Because the calculations to generate user reward status requires several reads of the Travel table in sequence, we want to ensure no other operations are occurring on those records between each action. Since it is possible for many users to trigger this procedure simultaneously it is important to avoid any possibility of them interfering with each other, where one might alter needed values before another completes. Updating the rank for 2 users simultaneously for example could throw off the quartile groupings, adding more members to a certain rank than should be allowed.

Another component of this database program is a trigger on the Travel table which is activated by any Insert operation on the table. Naturally this trigger is activated as soon as the above procedure creates an Insert event, but it will be blocked from its update or insert operations until the procedure's transaction is resolved completely. This allows us to accurately assess recipients of the King of the Hill/ "1 in a Brazilian" award, since that badge depends on multiple values that are being set by the procedure. When the procedure finishes the trigger is then able to apply these awards to riders who meet the criteria. Once this award is achieved it is retained forever, even if the user's status were to fall below the initial requirements.

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.

Angela: The most technical challenge in my opinion was loading the data from the text files into Google Cloud. While we were confident in our create table statements, none of us had ever loaded data into buckets to be held in the cloud. The tables that were used more in a reference way such as the calendar table we're able to be loaded in quite easily. A minor issue we encountered along the way was that when we declared the data types for each attribute was different from the header which was always a string this caused an issue loading in data. I was able to resolve this with a quick fix by simply removing the top row in the CSV file.

Once we had solved the issue with the headers as strings and the attributes being integers we still were facing issues with loading and data and its complete form. What we had not considered when we started loading in our data was the dependencies of the primary and foreign key relationships on the other tables. To say in a different way, if one table's attribute was referencing another it was important that the other table was loaded first. We started noticing a problem when we would load in larger tables and only a portion of the data would be loaded in.

We at first thought it was an issue between the CSV files and the Google Cloud, but upon further inspection we discovered that it was due to these dependencies of related tables.

So once we removed the tables and put them in an order that made sense with regards to the referencing attributes, we were able to load in all the data without problem.

Shon: In my opinion, one of the largest issues was the usage of website development tools. Nobody in our group had any background in web development, so we spent a lot of time trying to understand how it worked. This took an extraordinary amount of time, to the point where we started Stage 4 early and barely finished it on time.

We had many things we needed to figure out. First, we spent some time understanding the syntax and how to apply it to create a basic website. Afterward, we spent a lot of time understanding how to connect our database to the website itself. Given the limited resources we had, this took an exorbitant amount of time to understand and debug when implemented. leading to our rush for completion.

In the future, I recommend taking the time in the beginning to understand how web development works. It is extremely useful for not only this course, but creating future sites that require a database. Thus, taking the time to absorb and apply these new concepts will help new students succeed in the course and beyond.

Nate- One of the most time consuming challenges we faced was getting the order and declaration of our procedure, transaction, and trigger for the advanced database program to assign user awards. We initially wanted to trigger all of the updates, including the procedure execution, after any insert into the Travel table. It seemed this would naturally fit the function of a trigger, however we soon realized that it was not possible to call a stored procedure this way and had to rethink our approach.

While coding the stored procedure we had a good sense of what we wanted the procedure to do, and what isolation level we wanted for the transaction within, however we had some initial difficulty declaring the transaction correctly. While we knew the syntax from the mySQL documentation, we were unsure at which point in the code the transaction would need to be started. Through some trial and error, and use of Google, we eventually found that the transaction needed to be started after the declaration of the procedure variables, including the cursor.

We also initially thought our trigger would be on the Rewards table and for any insert operations, since the rainbow award is done via insert and is required for the award the trigger manages it would only need to run after an insert operation. However we found that this created a potential for infinite loops, where the trigger causes an insert, thus triggering itself. The database engine helpfully prevents this as well, but we had to reconsider our strategy. We decided that since awards are based on activity from the Travel table that we would locate the trigger there instead, to be activated by the insert operation of the stored procedure, but unable to attempt its operations until the procedure completes thanks to the transaction, ensuring the trigger performs calculations only after all of the final values are written to the table. This was definitely a learning experience but we came away with a much more complete understanding of how these database elements interact.

We have two main pieces of advice to offer for future projects.

Firstly, it is beneficial to use one of the “suggested” datasets given from the course material, unless you are very confident your chosen data set will meet the complexity requirements needed to get full credit on the project stages. We entertained several ideas we thought were interesting at first, but most datasets we found either had a small number of rows, or not enough relations to produce queries of the needed complexity. We also think this is beneficial because it requires that you become familiar with a new dataset, understand the properties of its relations, and how they fit together, which is a scenario you are very likely to encounter in the real world when doing research, consulting, or beginning work for a new employer.

Secondly we think it is *critical,* borderline necessary in fact, when creating a google cloud application to connect the cloud database and VM to locally hosted editors. The recitation for setting up GCP actually does a nice job of showcasing how to connect a local mySQL Workbench to the cloud SQL instance. For the application VM we would recommend connecting via SSH using VS Code or similar. Navigating the file system via the terminal is a good skill to know, and you will have to do it in order to initialize the VM and when starting it up. However once it is created, it is much more streamlined and user friendly to be able to navigate the file structure and various documents using a proper editor/SQL workbench. We feel this would save a lot of time, effort, and frustration if done correctly. If for no other reason that it makes navigation, copy/paste, and saving files much more intuitive and easy. Although there are some pitfalls to be aware of; you may need to reconnect the SSH instance if the connection goes “stale” or times out on the VM, it is necessary to stop and restart the VM instance via the browser based shell every time changes are committed via an external editor over SSH.

A helpful video describing the steps to SSH to a VM instance using VS code is included below. This is what we used, it takes about 10-20 minutes and saves *much* more time over the course of the project: SSH into Remote VM with VS Code | Tunneling into any cloud | GCP Demo

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

For the most part we have covered all the changes in other sections of this report, however we think it bears repeating that the most significant changes between our actual application and the original vision of the app in the proposal are:

I. We initially intended for users to actually be able to log in, and use session variables to maintain their ID for other operations within the application. We researched how to implement this, and believe it would be possible for us to accomplish, but were not able to pursue it due to time constraints and the need to focus on the core functionality of the app. This is likely the first improvement we would want to make to benefit user interaction with the app.

II. We had initially hoped to include some sort of pathing element between start and end points that would allow us to calculate statistics like total fare cost and approximate travel time for each trip. However the complexity of determining complete, let alone ideal, paths through the transit system was not feasible given the time we had and the demands of other critical app features. For now we are showing the user possible trips they can take given a destination, which they may search for, and the average “frequency” they can expect that trip to be available at the time and date they specify.

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

The first aspect that our group would like to improve upon had we more time would be the variable names on the back end. While we were debugging our transactions, stored procedures, and triggers we would often discover a new direction that we would like to go into but not change the name to reflect this new direction. For example we have a transaction called getUserReward, we also have a transaction called rewardUpdate, and a transaction we did not use just called reward. This naming convention is confusing when we want to reference code that we had written previously and now we have to sift through multiple iterations of similarly named functions.

After completing the relational algebra unit, I feel like the group and I would have a better idea on how to further optimize our queries especially the ones that are doing multiple joins and taking a second look and discovering a better order for inner joining the relations based on the size of each relation and the unique tuples. For example we have to join the Routes, Trips, Stops, and Frequencies table when we want to combine data and extract it in a meaningful way due to all of the foreign keys and the distribution of the data. The tables have varying amounts of tuples so making sure that we go from the smallest tables to the largest tables will help decrease the amount of time and further optimize our queries.

An aspect that would be interesting to look into had we more time would be to recommend rides to users. We could use a machine learning application to predict routes that might be useful for the user based on the routes they have already saved. For example we could take a look at the most visited places in Sao Paulo Brazil and verify to see if the user has visited those places and if not recommend a route from an origin that they frequently use to this iconic tourist destination. This would be something we could Implement on the back side. So based on whenever the insert happens into the travel table that a user saves we could identify frequently visited locations from other users and run a cursor to loop over and see and if not visited then we could return some places the user could be interested in visiting.

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

Stage 1 was generally well-distributed in terms of work, where everyone spent time researching different databases and coming up with some sort of project. Shon did most of the work in Stage 2. He created the entire ER diagram, logical schema, and assumption descriptions. Stage 3 was also relatively well distributed, with Shon writing code for the tables, Nathan creating the advanced database queries, and Angela creating the tables in the shell, uploaded data to the cloud, and analyzed the query results. For Stage 4, Nathan did a lot of the work in terms of understanding how the front-end works and connecting it to the back-end. Angela spent a lot of the time organizing the website with creating views and UI interactables, connecting frontend and backend with one of the advanced queries, and using similar tools to ensure everything worked. Shon was assisting both in the debugging process to ensure everything ran smoothly. Everyone worked together on every aspect of Stage 5, so it was all done at once. Lastly, Stage 6 is also equally distributed, with everyone answering an equal amount of questions. Given our ability to consistently meet, work, and learn together, the team was managed well.