Juhi Pathak (jp48969)
Maria Velez (mav2896)

# M6: Technical Report

We will describe the challenges we encountered on our final project by going through each milestone in order, and the obstacles we faced for each. For our final project we were provided with three different datasets containing information about music, such as artists, songs, releases and labels. Our goal was to create a data warehouse that unifies and utilizes entities from each of these three data sets. Over the course of this project we encountered many challenges to reach our weekly milestones. We cataloged and built upon each milestone, watching as our project grew into its final stage. We now reflect on the process and the obstacles that led us to the creation of our data warehouse.

The first part of our lab, after setting up our redshift and submitting our connection test, was postulating the layout for the schemas. We were given three imposing datasets to model and work with- Discogs, MusicBrainz, and Million Song. Some of these datasets, particularly MusicBrainz, were challenging at first since it was much larger than the datasets we'd used previously. Therefore it took a lot of time to understand the tables themselves, the information in each, how they related, and how we could use each attribute. With the end goal of writing queries in mind, our priority was thinking of which tables we would most like to implement along with which attributes within the tables we would like to use. Using LucidChart, we created a physical ERD model for each dataset. The most difficult dataset to deal with was

MusicBrainz, primarily because it has a lot of csv files that we had to sift through within the data subset. Despite the challenge of sifting through all the csv data in MusicBrainz, it allowed for a diverse schema and further helped when we had to create the unified schemas. When selecting tables for Discogs and Million Song, we kept in mind that we would need tables to merge with in our Unified Schema. Therefore, we included main tables such as releases and artists that were present in the other datasets. These 'merger' tables were also important for connecting to other descriptive tables such as MusicBrainz's genre table (which we use substantially in our queries) and lyrics from Million Song. When it came to reconciling the many to many relationships in the dataset, it was simple since the datasets contained quite a few junction tables- especially Discogs. One of the tables that confused us was the "Artist_Term" table in Million Song schema. It appeared that this table would be a junction table but there was no separate term table. We realized after looking through the data that the table contained the artist id and a term/genre/classification for the artist. We knew that we could create a separate term table and make the primary key term id and the secondary key the description, but it didn't make sense to do so at the moment since we weren't directly querying from that table. Asides from that, the junction tables were easy to understand. The unified diagram was a smoother process since we essentially were just replicating our previous three diagrams, with the exception of the cross-dataset relations. Creating the unified schema relationships was as easy as they were obvious. We knew that it would help to put relationships between a few tables as opposed to solely relating the artist tables from each dataset.

Therefore, we related the datasets on release tables as well. The unified schema was essential to make any further developments on the other milestones. It was also helpful to make the data dictionary, in the data dictionary, we had to log the purpose/function (e.g. joining, filtering, aggregating, grouping, etc.) of certain attributes from the tables in the unified schema; this helped us plan ahead on how we would translate our queries into SQL, specifically how we would write our join statements.

During this point in our lab, we both switched our lab partners due to irreconcilable differences. Fortunately, this didn't hinder any of our progress. We decided to use the team-f schemas and we formed a new group.

Once we established whose queries and schema we would use, our team moved on to generating tables within each schema by using a python generate DDL script as well as a script provided by MusicBrainz. These scripts took the csv file used and generated tables for us to use. This part yielded to be fairly simple. For the sake of organization, we put the created table files into separate folders, labeled by each schema/dataset. Next was created the schemas for each dataset. After setting the search path and create schema statement, we located the path for each of the files for the table that the schema would need. We continued the same process for the other two datasets. Running the files into the terminal that created the schemas went smoothly, and we were able to view the tables in each schema. But, it was the next steps that were more difficult.

After the tables and schemas were created, we needed to copy the data from the repository and load it into our tables. We created a file labeled copy_‘schema name’.sql that contains numerous copy statements formatted as:

“copy Artists from 's3://cs327e-final-project-datasets/discog-csv/artists.csv' iam_role' arn:aws:iam::943810861521:role /redshift_s3_role' region 'us-east-1' csv quote '"' ignoreheader 1 trimblanks compupdate ON maxerror 50;

When we created this type of file for each table and ran it, we came across several issues. First, we were getting an error with the MusicBrainz copy statements. The data wasn’t loading into the tables. Upon further debugging, we realized that there was a mismatch in columns in the tables created and the data that was to be loaded. Essentially, the DDL provided by MusicBrainz didn’t create every attribute for the table and when we tried to load more data than the table could hold, it gave us this error. We went back into our create table statements and had to account for these extra attributes by adding them in. Then we had to re run the create schema file since we changed some of the tables. After we ran the copy statements again, the data successfully loaded for some of the tables.

Second, we came across an with the Discogs copy data file. When we ran the copy_discogs.sql file into the terminal it would run, but when we ran select statements to view the data, the tables were empty. After meticulous scanning, we realized that our IAM role was incorrect. After fixing this, we re-ran the scripts, ran select statements, and we saw that it successfully loaded some of the data.

Lastly, some of our tables were not successfully loading, therefore we were required to debug the errors using a script written as “select line_number, colname, type, raw_field_value, err_reason from stl_load_errors order by starttime desc;”. When we ran the debugger, the first thing we noticed was how poorly the formatting was on the terminal; it was really hard to understand. The writing would run off the screen and we weren’t able to identify what needed to be corrected. After talking to professor Cohen, we realized that we could modify the debugging script so that it only shows us the error and not unnecessary error attributes that were taking up screen space in the terminal. Once we ran this modified script, the terminal displayed clearly what needed to be fixed.

For the most part, the errors were given due to the length restriction on some of the create table attributes; they were not big enough for some of the data that would be loaded. The error would read as “String length exceeds DDL length”. Therefore, a change in varchar length fixed most of the errors that the debugger was displaying. Another error came with some of the attributes being listed as having a date type. Because date type is a somewhat strict formatting, when we changed it to varchar it successfully loaded. We knew that it wasn’t useful to spend too much time figuring out how to format date since we would be fixing this issue in the next milestone.

As mentioned previously, creating the ERD’s in milestone 2 was very important considering we referenced it many times. On the Data Integration milestone we were tasked to identify which columns we would be using for cross dataset joins on the three schemas in order

to create a single unified schema. After glancing at our schemas, we determined that we'd be joining MusicBrainz and Discogs on their "Artists" table as well as their "Releases" tables. For Discogs and Millionsongs, we merged them on "Releases" and "Song Summary", respectively. Therefore, these were the tables that we would need to modify. Unfortunately for our team, we misunderstood these instructions and modified many of the attributes' varchar lengths and removed punctuation for many attributes that we didn't need to do it for. It was unnecessary extra work. Luckily, modifying the length of the varchar and the punctuation didn't negatively affect our queries, especially since we saved the modified data under a different title (i.e. artists vs cartists). One of the biggest problems we encountered during this milestone was retrieving the max bytes or running any sort of function in the terminal. It was taking an excessive amount of time to retrieve the maximum bytes. Our initial reaction was to make sure that the IP address was correct. Second, we checked to see if perhaps changing the region would affect it. We found that it didn't help. Lastly, we noticed that even when we were running it from our homes, it seemed to take forever. After speaking with Professor Cohen, she explained that we might need to reboot our Redshift role. Upon doing so, our commands started running at appropriate times. This was actually a big obstacle in that it took a lot of our time, particularly useful time during class when we could've asked the professor and TA's for help. However, with this issue we learned that you could have multiple terminals open and running different commands at the same time. This helped expedite the process for us, especially since the deadline was nearing.

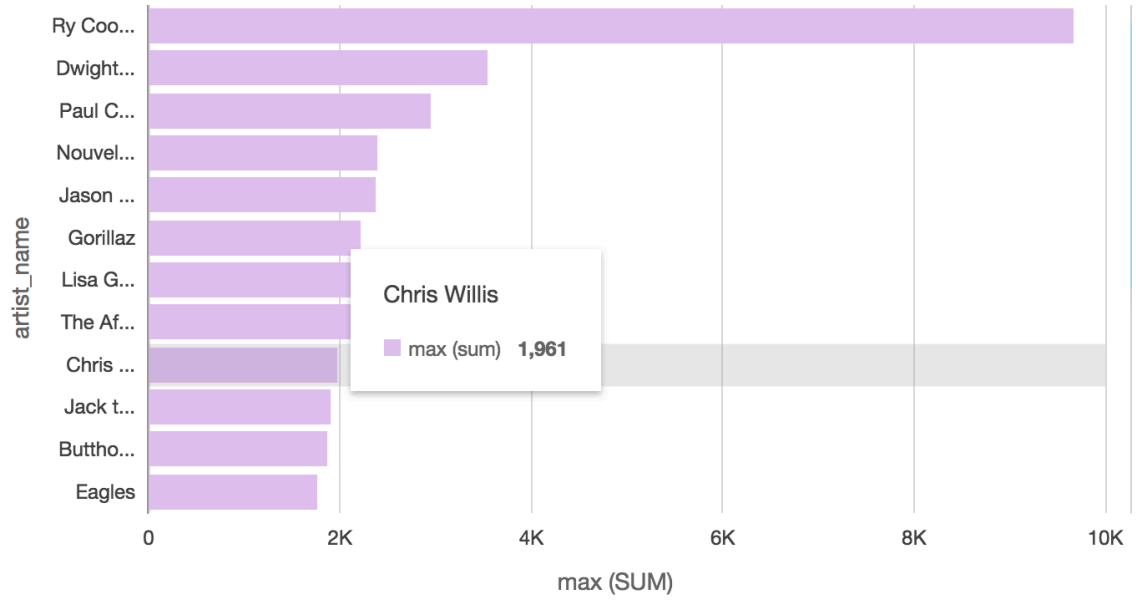
Once we successfully modified all the data, we created the unified schema. This part also took some time, since now we had to ration out which tables we actually used for our queries. Because of this, we also had to further think out how these queries would be written in SQL and on what tables we'd be joining. Thankfully, we referenced back to our data dictionary as well as our diagrams to accurately select which tables we'd be using. Once we filtered out which tables we'd use, we set the search path as unified, and wrote all the create table statements into the create_unified.sql, we ran the SQL script into the terminal. Immediately, we ran into errors, almost all of which were syntax. We had some duplicated tables, some of the create table statements had attributes that weren't referencing anything and some of the table names that were being referenced had different names in our Redshift than in the script we were running. Once we fixed these errors and re-ran the code, it seemed to have created the tables successfully. We ran a few select statements to confirm that the table and data was there, and fortunately it was.

The last part of this milestone was to translate our queries from plain text into SQL code. Surprisingly, this didn't take much effort or time out of all the things we've done so far. The first half of the semester, which included a test on SQL query writing, prepared us well for this part. We split up the queries in half: six queries for each of us. While writing our queries and running them, we found that some of the queries didn't yield great results. For instance, one of our queries was "Which labels released the most pop songs?." When running this SQL code, we limited to the top three labels, but one of the labels contained "No Name" as the

name, which meant that there must be some association, but the name itself was missing. This made our search result look less pleasant. We then changed the query to ask which labels released the most classical releases, which yielded actual label names. Another one of our queries was, “Which Children’s songs use the word hate”. When we ran this query, a few songs came up. And after searching the songs, they didn’t appear to be children’s songs. We tried the same query, but instead we used the category “spiritual” songs. This gave us more songs and songs that, when searched on Google, appeared to be classified as spiritual or religious. Once completed, we ran all the SQL statements at once to confirm that the queries correctly selected the information that was needed.

The final part of the assignment was visualizing our data and queries using Amazon’s QuickSight tool. After setting up our QuickSight account and information we created views for each query we had written and saved it as create_views.sql. We set the timer and run the sql script into our terminal. Once the views were created, we timed each one. They ran smoothly initially, with the exception of one. We realized this was because our console had frozen. We ran it again and the times on the views were all luckily very fast; meaning we didn’t have to change them. When it came time to displaying the views onto QuickSight, we had to modify and even completely change some of our queries so that they would visualize better. A lot of our queries used the SQL count command. When we displayed these particular queries onto QuickSight, the visualization would be just bar on a bar chart that had the value of the count. For some queries we used the max function and the QuickSight result was a just a one number. For example, the QuickSight visualization for the following query, which finds the maximum

number of times one user played the song “Baby”, only resulted in a big “152”: SELECT max(p.play_count) FROM millionsongs_songs_popularity p JOIN millionsongs_songs_summary s ON s.song_id=p.song_id WHERE s.title='Baby'. These results looked very uninteresting. Therefore, we changed our queries to select titles or names and their respective counts and ordering by the counts. In other words, we made sure we had two columns with multiple rows. When modeled afterwards, we had better, diverse and interesting visuals. For example, the query that yielded simply a “152”, we changed into another query that answers the following question: what's the maximum number of times somebody has played a song by each artist (songs played more than 1500 times). This yielded a much more interesting visualization:




One of the most interesting parts of this project was how many times we had to change our queries through each milestone. Initially, we made our queries and we kept in mind that we would need to use both data sets. For the sake of options, we created thirteen queries instead of the required ten. We knew from the past labs that when it came time to write our queries in

SQL, some would be more difficult to execute. Therefore, by having more options, we could choose better, more diverse or simple easier queries to execute.

When it came time to do the actual SQL writing part in milestone 4, we realized just how messy the data sets were. Many of the songs or artists were missing certain attributes, which we needed to query. Therefore, we had to modify the queries so that the results would show better - as opposed to having some of the results just be “no name” with a count written beside it.

During the QuickSight, we realized we made a mistake because we weren’t really looking ahead to milestone 5. Like previously stated, when it came down to visualizing the data, many of our queries used count statements or max, which didn’t translate well onto the QuickSight visuals. Because of this, we ended up changing some aspect in most of our queries.

If there were one thing we’d improve, it would be our planning process. Certainly, we were given milestones that acted as our “planning process”. But if we had looked ahead at what the future milestones contained that would’ve helped us immensely as we would’ve saved ourselves some time we spent correcting mistakes that showed up along the way. For example, we wouldn’t have wasted as much time as we did modifying our queries. Furthermore, we wouldn’t have wasted the time that we did trying to set the accurate varchars in the create tables during Milestone 3 since we ended up using the “get bytes” function later in Milestone 4. There were also times where we misinterpreted the instructions. For instance, in milestone 4, instead of modifying just the attributes that we would do a cross dataset join on,

we modified (removed punctuation, capitalized, etc.) almost all the tables in Discogs and MusicBrainz. When we got to Million Song we realized that this was completely unnecessary and was indeed a waste of time.

Overall this final project was educational and challenging in many ways. Learning how to use Amazon Web Services, Redshift, and Quicksight will provide to be useful in our future endeavors. Furthermore, learning SQL will be a great skill to have if our careers lead us in the path of handling large sets of data. We were informed going into this that our dataset would be very messy and, at times, inconvenient. With that in mind, we acknowledged that some of our issues would just be a matter of adapting to the database. Doing things, such as modifying our queries and not using certain attributes, allowed for us to clean up this data and streamline our querying process. Many of the errors we encountered helped us grow stronger in our skills and more patient in our debugging. With the help of our peers, TA's, the internet, and Professor Cohen, most of our issues were solved. In conclusion, the project had its challenges but was successful overall. There is still much to be explored and we look forward to see what this newfound knowledge inspires us to do in the future.