Pablo Rivera
pr8596
CS 327E

Milestone 7: Final Report

In our final project, we were tasked with creating a data warehouse over contemporary popular music. We were given datasets drawn from Discogs, Million Song Dataset, and Music Brainz to work with and populate our data warehouse. The overall goal of this project was to put our knowledge of SQL and database relationships to the test so that we could successfully complete each milestone set upon us. We used Amazon Redshift to manage and store the data and LucidChart to create the schema for our database.

Database Design

The first major milestone we came across was to design schemas over the datasets that we were going to use. As we had learned in class, the best way to ensure that your database will work correctly is create schemas in which we can visualize how the data is connected. Jacob and I began to look over the files in the datasets and began to determine which files we wanted to use. Of the three datasets to work with, we chose to use all the files from Discog and about half of the files from Music Brainz. The reason our group chose to exclude from working with the Million Song Dataset was because we felt that a lot of the data in the dataset was similar in nature to that of the extensive Music Brainz data, and so we believed it would be useful to prevent ourselves from working with similar data. Once Jacob and I decided upon which specific files to use, we had to create schemas in LucidChart that detailed how the each of the datasets were connected. This step was crucial in the team's efforts to build the data warehouse

because we had to be confident in our understanding of how the relationships in the datasets were set up. Once we created the schemas for the Discog and Music Brainz files, we needed to create the design for the unified database. Upon reaching a deep understanding of the datasets worked, our group decided to create junction tables for the Artist, Release, and Label files that Discog and Music Brainz each contained. With the creation of just those three files, we could navigate through the relationships of over thirty different files. Once the schema design was finished, we created a data dictionary for the unified database via Microsoft Excel and we came up with a list of ten queries that we would like to implement in our unified database.

Data Load

The next milestone that needed to be completed was to create the schemas via Amazon Redshift and load the csv tables into them. To help with the tedious task of creating multiple create table statements by hand, we were given the python script generate_DDL.py that takes in a whole CSV file and produces an SQL file with the create statement. Our group used the provided python script to generate the create table statements for all the files in the Discog dataset without any setbacks or difficulties. The class was also provided with the SQL script to create the necessary commands to create the tables for the Music Brainz dataset; However, we were warned that some of the constraints in the provided script would not be compatible with Redshift. We found that two of the Music Brainz files had issues with the provided create table statements. The files in question were the *Track* and *Release_Group_Alias* files.

Our group stumbled upon the issues when we attempted to copy the data from the S3 buckets and into our own Redshift instances. The problem with the *Track* file was that "name" column for some of the rows had too many characters and so could not be copied successfully. To get around it, we first attempted to set the column's type as text because we believed that text-types allowed any size for the string. To our dismay, the idea did not prove successful. The team eventually just settled with trying different sized varchar numbers until the data could copy without any issues. The size of the variable type that worked was a varchar of size 2000.

For the *Release_Group_Alias* file, the issue arose from the fact that we made our create table statement too short. While we only created the columns that we wanted to use for the database, the COPY command in Redshift transfers everything in the table from S3. To combat this, I appended 10 dummy columns at the end of the create table statement. This allowed us to copy the entire table over without any issues.

Data Integration

In this milestone, the team was charged with creating and populating the unified schema within our Redshift cluster. The unified schema's tables and respective columns would be derived from the Discog and Music Brainz datasets, but the data needed to be cleaned up first. To do this, we were given an SQL function called *get_utf8_bytes* that we added to our Redshift instance. This function then determined the size of the largest string in a given table's column. We used this to append column with a varchar of the max size to tables that contained useful data. The new columns were then trimmed

down with special SQL function that standardized the data by removing punctuation marks and unnecessary extra spaces. This part of the process was relatively simple and issue-free. However, it should be noted that this step took a considerable amount of time because user-defined functions are usually significantly slower than built-in function in Redshift. One of the primary challenges was to clean and sort the date type column into uniform entries. However, our team did not need any dates for our queries, and so we did not need to perform that extra step for our data. Once all the needed tables had their data standardized, we populated the unified schema by copying over the needed tables and columns with simple create-table-as-select statements. Since the data was already refined and standardized, we did not run into any issues populating the unified schema.

The final step in this milestone was to review our queries and refine them to create legitimate SQL queries suitable for analysis in the unified schema. Our group had to change most our queries because we mainly created single table queries. However, after looking over our unified schema database design, our team created queries that crossed multiple tables and established relationships across Discog and Music Brainz data. We were able to do this by using multiple JOIN commands in our SQL queries and making use of the *Artist_Join*, *Release_Join*, and *Label_Join* junction tables that connected the different datasets. As noted earlier, this was because we created detailed database schemas in LucidChart that enabled us to visualize the connections and anticipate the needed junction tables.

Pablo Rivera
pr8596
CS 327E

Quicksight Visualizations

The final milestone had two major steps to it - the creation of virtual views from the queries and then Quicksight visualizations for those views. The creation of the virtual views from the queries was not a difficult technical feat to accomplish because views are made with simple *create view as…* statements. The requirement was to keep the runtime of the view to under thirty seconds. If the view failed to run in less than the allotted time, then we were to attempt to optimize the view by attempting to pre-computing as much of the view as possible. Our team's longest running view was only a few seconds long and so we did not feel the need to pre-compute any of the queries.

The second step in the milestone was to create the Quicksight visualizations. This was done by selecting the views in the Analysis page and playing with the visualization of the data. Our team encountered troubles with this step because a portion of our queries and their subsequent views displayed scalar values or values in a single column. Obviously, this makes it quite hard to play with nice visuals when you only have one data point. Thus, we had to manipulate our queries and their respective views and adjust them so that they display data as columns. For instance, instead of having a query that returned the number of releases under the artist Kanye West, we changed the query to display the number of releases per different labels that Mr. West has worked with. Once the data was in the correct form, the visualizations were much easier to display and manipulate.

Pablo Rivera
pr8596
CS 327E

Reflection on Problems

While the team was successful in accomplishing the milestone with relative ease, there were a few instances in which we were inefficient with our work. For instance, we encountered a couple of issues with copying the data into our Redshift clusters. The main problem in that stage was that our group did not know how to determine the size of the varchar needed to create and copy the "name" column in the *Track* table. Our team recognizes that our method of guessing large numbers until the table copied successfully was wildly inefficient. We now understand that it would have been much simpler to simply create a script that would determine the size of the largest string the in the column in question. This function could have worked like how the user-defined function *get_utf8_bytes* worked. However, this could easily become a double-edged sword because while that would have saved some of the time we spent trying to determine the size of the varchar, the function we could have created might have been extremely slow since it must parse through the entirety of the massive *Track* file. Despite this potential pitfall, we still believe that it could have been a viable option when compared to our guessing game.

Improvements to Be Made

One major improvement to our demo that could be done is related to the Quicksight visualizations that we made. We essentially created different bar graphs across all our data. In hindsight, it would have been much more interesting and visually

appealing to view the data in different forms such scatterplots or pivot tables. We could have remedied this issue by attempting to play with the Quicksight application more and determining which visualizations were feasible given our data. Another idea that we came up with was to create different queries. Most our queries and their subsequent virtual views were displaying columns composed of a categorical variable and a matching quantitative variable like an aggregate count. We should have attempted to switch up the type of data that we took as our output and perhaps take multiple columns of different types of variables. This would have resulted in a lot more variety in our submitted Quicksight visualizations.

Another major issue we came across was the elimination of unique names for artists, tracks, labels, etc. when we did the standardization of the data. When we removed most of the punctuation and chopped down the strings, we came across some instances in which artists could have same name in the unified schema despite the fact they are unique in the original datasets. As of now, we have yet to determine a method in which we could eliminate this issue from arising again.

Lessons Learned

This project was a culmination of everything we had learned in class, as well as new material that we came upon as we moved through the milestones. This project was a great lesson in where we learn to apply the techniques we've learned and build a data warehouse from scratch. For instance, we were honestly quite flustered with the fact

that we needed to create schemas in LucidChart for each dataset used and for the unified schema. We originally felt that this was a waste of time but we have learned the errors of our ways. We have come to understand that great schema design helps eliminate a lot of future problems. If we had not done the schema designs, we would not have learned that we needed to create junction tables that tie our datasets together. Without those junction tables, the datasets are separated and our unified schema and its queries are not connected and therefore worthless.

Overall, this project was a great experience for the members of our team. We both are looking to work as data scientists and this project has enabled us to get our hands dirty by working with real world datasets. We enjoyed working on this assignment and hope to see it positively affect us in our search for satisfying careers.