Elias Ansari (eaa957)
Khalid Hamza (kfh294)
The University of Texas at Austin
CS 327E Elements of Databases
Milestone 7: Final Project Paper

April 24, 2017

Abstract

This paper describes, explains, and analyzes the process our group, recurscursive, took to build a data warehouse. The data warehouse was used to collect information about popular music from three different data sets: Discogs, Millionsong, and Musicbrainz. The goal of this project was to create a unified schema that joins the unique attributes from each data set to discover interesting details of the data our group put together and analyzed. Our team analyzed a variety of different characteristics of popular music, a few examples being the distribution of genres, the artists with the most releases, and the most common formats of releases.

In this paper, we are most interested in analyzing the process of creating a data warehouse to hold information about popular music. As the class learned in the beginning of the semester, the process in making databases is crucial for long-term functionality. This project emphasizes a hands-on approach to this process by allowing students to create a functional and structurally-sound data warehouse while collaborating with other students. By working with peers, students are not only learning how to develop a data warehouse to store information, but they are also learning how to communicate logic, code, and ideas to others, which is an important skill to have in the workplace. Throughout the semester, our team used a variety of software, languages, and programs, including: Amazon Web Services (AWS), LucidChart, SQL,

the psql shell, Python, Amazon QuickSight, Git, and Github. All these different tools will be explained in further detail later on, however, it is important to understand the variety of tools used throughout this project.

Outline

Before delving into the details of the technical aspects of the project, the general outline of the paper will follow a similar order to the outline of the milestones of the project throughout the semester. We will explain: the different tools used throughout the project, the environment setup, the database design, the data loading process, the data integration process, QuickSight visualizations, and finally the challenges we faced throughout the project.

Project Tool Descriptions

Amazon Web Services, AWS, is a cloud platform that all teams in the class used to store their databases. However, there are different tools within AWS which were used for different purposes, such as: Amazon Redshift, which was used for the data warehouse to analyze our team's data using SQL; S3, which was used to store the raw data for each data set our team used, where each data set was stored in separate S3 buckets held by Professor Cohen; EC2, which is a web service on AWS that uses the cloud to create a virtual computing environment; and Amazon QuickSight, which was used to to produce and present interactive visualizations of the queries our team created. LucidChart, a tool outside of AWS, was used for creating the physical and conceptual diagrams when designing the database. The different languages used throughout the project included: SQL, in order to actually create the databases and queries; Python, which used a script to input a CSV file from one of the three data sets our team used in order to produce an SQL file which contains the different create table statements (in other words, our team used

Python to create our tables in SQL); and the psql shell which was used to run SQL code to create the databases, schema, etc along with connecting to our team's Amazon Redshift cluster. Finally, Git was used for version control throughout the project.

Environment Setup

The first official step of the project was to setup the environment so that our group was able to connect to our Amazon Redshift cluster from our local psql clients. This was essential for completing the project, because if our team was not able to connect to the Redshift from our psql client, we would not be able to create the tables, databases, or schemas. Along with verifying that we were able to connect the cluster to the psql shell, each team member needed to create a Stache entry. This stache entry contained the credentials of the AWS account along with the Redshift cluster credentials so that the TAs were able to access the data and grade each portion of the project. In addition to the AWS setup, it was necessary to create a new folder and files using the precise naming convention given. One issue faced with the environment setup was the necessity to change the allowed IP addresses for our cluster as we changed locations or computers. This issue was resolved by reading the IAM "reference policies element" and the "authorizing access to an instance" documentation. For the purpose of building and testing we used the unsecure method of setting "::/0", which is equivalent to "0.0.0.0/0" as the IP address. However, this was eventually changed to a more secure method of authorizing only specific IP addresses and ranges. Using a conditional block within our element reference allowed us to choose several different ranges of IP addresses such as "192.0.4.0 to 192.0.4.255", thus allowing us to be able to access the clusters without the need to change the permissions every time we changed location.

Database Design

The second official step of the process involved designing the database. Students were given three datasets to choose from: Musicbrainz, Millionsong, and Discogs. Each group had to choose a minimum of 30 files using at least two of the data sets. Our team decided to use all three data sets with around 31 files to work with. After deciding the 31 entities from all three data sets available to us, we selected attributes from these entities that our team wanted to work with and created our data dictionary. The data dictionary is an excel sheet which documented the following: all of the 31 entities, the attributes our team wanted to use within each entity, the data types of each attribute, what we re-named each attribute, and any of the primary/foreign keys within that entity. After creating the data dictionary, we created 12 queries to cover all 31 entities we chose. It is important to note that these original queries were modified and/or removed as we advanced through the project to create more interesting and relevant queries. At this point of the process we used the data dictionary and our queries resources to create the physical diagrams for each data set using LucidChart. After creating these three individual physical diagrams, we created a normalized physical diagram for a single unified schema consisting of the chosen elements from the three datasets. The normalized physical diagram aggregated the different entities from all three of the data sets into one combined physical diagram, joining the different entities on the similar attributes they shared. In other words, the normalized physical diagram structurally and functionally combined the three different physical diagrams which we created beforehand. Lastly, after creating all of the physical diagrams, which totaled to four diagrams, we created our normalized conceptual diagram. This diagram differed from the normalized physical diagram in that it only contained the entity title and the attributes we wanted to use for each entity. The physical diagram included the entity name, whether an attribute is a primary key (PK) and/or a foreign key (FK) the attributes, the attribute name, and the datatype of each attribute. The database design portion of this project was one of the most longest parts of the project. Spending extra time ensuring proper database design is crucial and can save considerable time in later steps.

Data Load

The purpose of this stage of the project was to create the tables required and to load the data from the CSV files into those tables. Creating these tables can be done with a series of hand-written create table statements. However, because each data set contained a large number of tables, we used an automated process using a Python script called ‘generate_DDL.py’ to generate these statements. This Python script took in a CSV file as the input and produced a SQL file containing all the create table statements as the output. It is important to note that we only created tables for the tables we were using in our project and not all of the tables within the datasets. For organizational purposes, we created a separate schema (contained in a DDL script) for each data set and ultimately ended up with three DDL scripts. In these DDL scripts, we wrote code to drop the schema if it already exists, create the schema, and then create all the tables we wanted to include from each data set. After running these DDL scripts it was critical to check the datatypes that the automated script wrote. Since Musicbrainz is open sourced, it was possible to check the create table statements generated from the generate_DDL script to the create table statements from their public github repository. However, for both Millionsongs and Discog, we needed to go and spot-check the data types in the dataset and ensure that the they were able to properly be created and loaded. This process would continually repeat until there were fewer than 50 errors. The biggest issue with loading the data came from the Millionsongs metadata. This series of files contained a great deal of columns and rows. This subset took ~one hour to run each time, so the method of reloading the data and checking for errors was very tedious and took a very extensive amount of time. One method we used to speed up this process was to run a script to save the results of the attempted load of the millionsongs metadata into an “issues.txt” file which was populated from the stl_load_errors system table. The script was run with a descending tag as shown on the timestamps “order by starttime desc” to allow the results from the last attempted loading of the metadata to be shown first. This process was repeated until the errors were minimized.

Data Integration

The purpose of this stage of the project was to create a single unified schema that consisted of the normalized data from the three individual schemas. The first portion of this stage deals with transforming the data across the data sets into a single consistent format. First, all punctuation and special characters were to be removed from columns that have cross-dataset joins. These cross-dataset columns can be seen using a combination of the analysis queries and the normalized unified physical and conceptual diagrams from the database design phase. After determining the columns that must be normalized, we began to randomly sample the data from the individual columns in their respect schemas to find any punctuation. We made note of any columns containing punctuation and saw the way that punctuation use varied across different columns. For instance, within the column “title”, there were a good amount of strings containing parenthesis as part of the name of a song or album. However, for another column “name” punctuation was more sporadic and could serve as a replacement for a letter within the name. For this reason we decided that we would make one punctuation removal script, that would remove punctuation without a replacement, and run that script for all of the different columns across all the different data sets. The reason is that while this may invalidate a few corner cases such as an artist named “A$AP ROCKY” becoming “AAP ROCKY”, that incorrect name would be consist across the different datasets, making queries about that artist, valid.

The second part of the data transformation process involved changing the VARCHAR values from the imprecise values created from the “generate_DDL” script to their correct maximum value. This was done by running a python script provided within the assignment. However, after attempting to run this script for several CSV files, we determined that the runtime for this script was very extensive. This is due to python inherently have a much slower runtime than a structural queried language used in Postgres and Redshift. The script took over an hour with a gigabit wired connection for one of the tables that had around seventeen million rows. We allowed this script to run for over eighty minutes for a table “Recording” from Musicbrainz that contains over forty eight million rows. However, we determined a much quicker option. Rather than running this python script, we dropped the table from the schema and recreated and reloaded the tables using a rudimentary binary search method. We began by creating the table and using the value VARCHAR(4000) and reloaded the data; The runtime for this was around a minute. When this data properly loaded, we dropped the table and recreated the table using the value VARCHAR(2000). This operation repeated. We would use the value halfway in between the last used value if the load was successful and we would use the value halfway in between the zero and the last used value if the load was unsuccessful. The order of numbers tried is shown below:

4000 → 2000 → 1000 → 1500 → 1750 → 1875 →1937 →1906 →1890 →1898 → 1902 → 1904 → (1905 failed).

While this series of operations may seem tedious, this whole process took roughly 15 minutes, a fraction of the time that the runtime of the python script took. This allowed us to get the values of all of the VARCHAR elements. After finding the VARCHAR lengths we ran a simple alter table script and replaced the various VARCHAR datatypes to the correct length.

The next portion of this stage dealt with normalizing the format of date data types. This would mean converting any currently existing VARCHAR elements that represented a date. Our group did not have any cross-dataset joins that involved any date or year columns. So for the purpose of the assignment, this section was skipped.

The final part of the data integration stage was converting the queries previously written in the database design state into functional SQL queries. Some constraints with these queries required that at least a portion of these queries must cross two or more datasets. Additionally, these queries should provide interesting results that will be used for reporting views in the next stage of the project. It is important to note that these queries were changed fairly significantly in this portion of the project. During this step we changed the queries to ask much more interesting questions that could be better visualized in the next portion of the lab. Our initial queries had many queries that used the same tables or had very similar things that they were looking for. Many of our queries asked very simple things like the specific location of a certain artist or label. For the new queries, we focused on things like the top five artists who met a number of parameters or questions about different distributions such as gender and location. After all of the changes were made to the queries, it was necessary to go back to the data dictionary and the physical diagrams in order to make sure that the different forms of documentation were consist.

Overall, this portion of the project by far took the longest amount of time. However, using multiple datasets and learning to transform was possibly the most useful portion of this project.

QuickSight Visualizations

This final portion of the project combines all the different parts that we have completed so far and makes a virtual view showing the visualization of the analysis queries being answered. The first thing we did was write 10 create views statements consisting of the 10 most interesting queries. We chose queries that had interesting results that would also be visually appealing. We ran all of the create views followed by a simple query that would return all of the elements from the view. Many of our queries included limits, so we didn't have an issue with any of our queries taking longer than the allotted 30 seconds.

For the purpose of visualization, we used Amazon QuickSight. QuickSight was able to seamlessly gather the data from the redshift cluster using an auto-discover feature. This made it very simple to gather the data we needed in one place. Additionally, QuickSight offers a feature that allows you to select two or more columns and it will automatically generate a visualization for the data. Amazon QuickSight works very well alongside Amazon Redshift and made the visualization part of the project very simple.

Conclusion

Each portion of the final project allowed us to face different challenges and solve different problems in our own way. The datasets were, for lack of a better word, messy. They had entries that were out of place, null values, and typos. Additionally, the issue that we faced with the runtime of the python script required us to determine an unconventional approach to finding the solution. Working with various datasets and normalizing them was a messy process, however, this process is reflective of what we may experience in industry. The overall experience of this semester-long project was rewarding in both technical and non-technical facets.