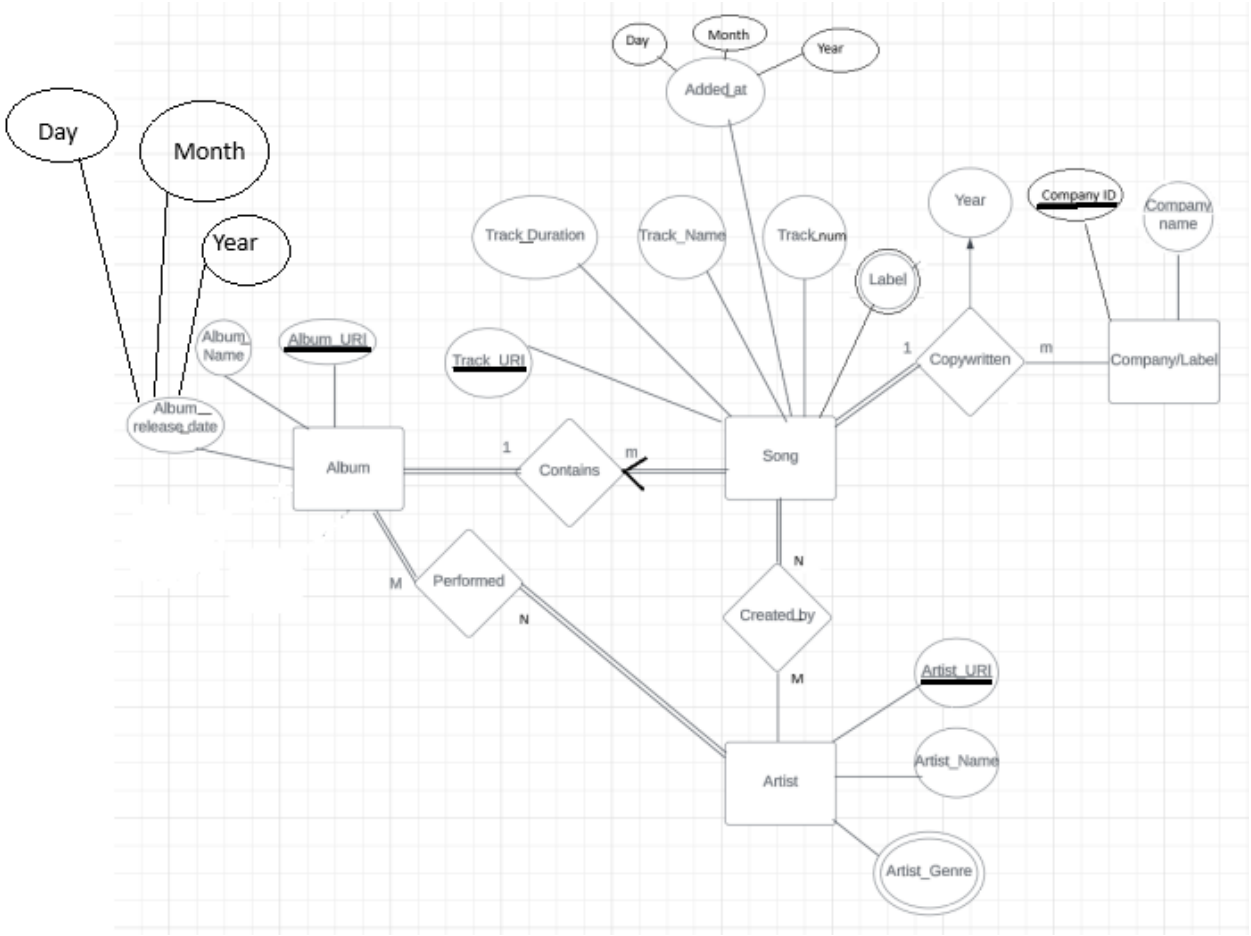
Databases Group Project Part 3, Project Report

The primary reason for choosing our particular data was that there were relationships between the columns that were immediately apparent. Such as artists producing songs, where each song belonged to an album, and where multiple artists participated in an album. The data itself consists of a collection of 10000 of the most popular songs from the year 1960 till the present, that being August of 2023, where the data was created based on rankings from the Australian Recording Industry Association, as well as the Billboard charts. The actual information itself consists of rows, where each row contained the songs URI and name, along with the names and URI's of potentially multiple artists or a single artist that were a part of the song. Along with that, each row also had the name and URI of the album that said song was a part of, and the track number pertaining to said album. The duration of the song in milliseconds is another column, along with the genres associated with the artists that participated in the song and the label that the song was signed to along with the copyrights of said song. In total, there are 10000 rows of songs along with the aforementioned data that accompanies each song.



For the tables Song, Album, and Artists the reasoning for breaking them apart into tables like that is straightforward. Each entry in the Songs table is the details pertaining to a single song, with that storing the URI, name, and duration of a song. Albums stores all information about a single album, which is the URI of said album, and the name. Artists stores the information of a single Artist, with their URI and name stored per row. Each one of these deserves to have their own table as they all are unique items that should be separated. Now for the other tables, the reasons become slightly more complicated, with Artists_Track being required since any song can have multiple artists who worked on it, so a separate table is required that stores the relationships between potentially multiple artists and songs. ArtistGenre is also required for a similar reason, that being since artists have multiple genres associated with them another table is required to break up the multi valued attribute, and store the multiple genres associated with any artist. Album_Artists has an identical reason since an album contains multiple songs, so a table is needed to break apart the list of songs that belong to an album into a table where each row states a song and the album it belongs to. Track_Copyright is also the same problem, where a songs copyright can have multiple companies listed, and Labels is the same thing as well since a song can have multiple labels. CopyRight exists because we wanted to assign each unique company its own ID instead of relying on the company name, and Album_Release_Date exists to break apart a compound attribute into its own table.

When designing our model, we didn't really run into any difficulties. Creating an ER diagram for the data was straightforward with the entities having natural relationships between each other. Many artists produce many songs, and a song can have many artists that produced it. A song belongs to a singular album, so that's a many to one relationship, as there can be multiple songs per album. An artist produces many albums, and albums can have many artists so an apparent many to many relationship lays there. For the copyright, we noticed that a song can have multiple companies that copyrighted it, so a single song has multiple companies that copyrighted it, resulting in an apparent 1 to many relationship.

The data model fit well into the relational database, with each specific 'item' getting its own table as expected, as well as each relationship between the items also requiring another table when a many to one relationship occurred.

Overall, few regrets were had concerning the decisions made in our model. Some issues were encountered, and some minor changes did occur. Those being that we did not include Album_image_URL in part 2, as it did not provide anything interesting to querying and just sat there in

each column as a useless piece of information. Some other minor changes included the merging of Album_Release_Date with Album as it was more convenient in the program, as well as the merging of Track_Added_On with Songs. A major change we had to make was not including any information concerning copyrights and the companies associated with the copyright. The reason being that in the raw data, each copyright was stored seemingly with random syntax and with various symbols in between that prevented any parsing to work on all the copyrights, which would have resulted in lots of redundant companies to be created and inserted into the database and would not provide meaningful information.

We would argue that the way we have modelled our particular data matches the data itself very well, and any other modelling of the data would be more complex or convoluted than it could potentially be. We struggle to see how you could model Artists creating songs, albums containing songs, and albums having artists in another fashion. The model is already straightforward enough to the point that we can't imagine another useful model for the given dataset.

Some interesting queries from our database are as follows:

collaborators <name> - This query returns the names of all the artists that have worked together with the artist with the given <name> on a song. For us, this query is interesting because it required us to think a bit about how to select only the artists that have worked on an identical song with a specified artist. In SQL, we ended up starting with a list of Artists with all the songs they have worked on, then joining that list with all entries of ArtistsSong such that the songs were identical, but the names weren't so we ended up with a list of artists and their collaborators, which then could be filtered to check for the collaborators of a specific artist. All in all, we had to think a bit about this one and it results in a satisfying query.

topTenAlbumsMostArtists – This query returns the names of the ten albums who have the most artists who worked on it, along with the number beside the name. This query was interesting because we needed to use aggregate functions and group by Albums UID in order to see the total number of unique people per album, then display the top ten. It's intriguing to see which albums had the greatest number of people who worked on it, and surprising to see that the top albums had more than 20 people who worked on them.

topTenArtistsMostGenres – This query returns the names of the artists with the most genres associated with them, including the number of genres beside the name. Using an aggregate function to count the number of genres associated with each artist and then grouping by artist UID resulted in an interesting list of artists who had lots of genres, which then could be queried with another command to see all the genres that could possibly associated with a single artist, a fun query to have.

Our database doesn't necessarily need a relational database, but it is incredibly well suited for one, so other database systems wouldn't really be able to do a better job in modelling our data. The only real other option we could consider is using a Neo4j database and modelling each of the entities as nodes, and the relationships between them would be natural as well. A Document database wouldn't really be suited here, and with the many relationships between Songs, Albums, and Artists you would be recreating an SQL type database with lists and dictionaries which would be pointless. The interesting queries we chose could be implemented in about the same level of difficulty to implement in Neo4j, and we could add some other interesting queries in Neo4j that would allow for example looking for artists N number of degrees away from a specified one. Any document database would just make any of our queries just as difficult or more difficult to execute, as we would just be replicating a relational database with lists.

We think that this database overall could provide some usefulness as a teaching tool for COMP 3380, but it definitely can't be the only one used. It provides clear and straightforward problems for students to solve and could serve as a good first database to create, with the ER diagram being straightforward enough to create, and the normalization requiring steps 0NF to 3NF also supplies enough of a challenge. The queries that can be created also can be fairly interesting, such as trying to get all artists who've worked together with a specified artist. In total, this database could serve as a good teaching tool for students in COMP 3380 and would be useful as the first database they could work on their own, providing clear relationships between the entities and has enough problems for students to get a good understanding of what to do correctly for a more complicated database.

Summary of Contributions:

Part 1:

(A) Data Discovery: We both looked around for datasets on the internet, and eventually together settled on the top 10000 songs on Spotify, only after both of us looked at multiple terrible datasets, such as datasets for crime reports or school information.

(B) Database Design: We both designed the ER diagram and iterated upon each others' ideas multiple times in order to come up with the end product. Leon initially converted the ER diagram into a relational model and normalized it roughly to BCNF. Krupal assisted during the normalization when needed and polished it up at the end.

Part 1 Final Deliverable: We both worked and polished up the final deliverable, with Leon working primarily with the summary of the dataset and writing down the steps of normalization that was worked on by both of us. Krupal worked on drawing a polished version of the ER diagram and listed the justifications for participation and cardinality constraints that we both came up with together.

Part 2:

We both worked on the java program that read in the lines from the csv file that contained the data and converted that into SQL commands, and both came up with the schema used. Krupal got the connection working for the database, and Leon designed the commands that could be run in the interface. In the interface, we both worked on command line menu terminal, and roughly split the work for the 18 queries assisting each other whenever needed.

Part 3:

Leon wrote about the summary of the data and the interface, Krupal wrote about the database, and both of us worked together on the discussion of the data model. Krupal wrote about the first 2 interesting queries and Leon wrote about the 3rd one. We wrote about the other interesting discussions or summaries together and wrote the summary of contributions together. Overall, after writing everything we still checked over and edited things all together.

In summary, Krupal and Leon contributed an equal amount of work to the project. Our combined efforts were substantial, and we supported each other effectively during times of need.

References

Top 10000 Songs on Spotify 1960-Now. (2023, July). Kaggle. Retrieved October 15, 2023, from https://www.kaggle.com/datasets/joebeachcapital/top-10000-spotify-songs-1960-now