Matthew Hileman
CS4420-001 Database Design
Spring 2022
Professor Brouillette
Music Chart Database Report
Project Due: 2 May 2022

Initializing the Database in SQL

After importing the top100 table into MySQL using the given instructions, a 3NF database needed to be derived from unnormalized top100. The very first table created was ***people***, which is a combination of artists and composers to first rid of the repeating composers group - but also standardize everyone into one single group. Artists can be separated from composers within the people table from a foreign key that will exist in a table track, which is created later. There are only two attributes in people: person, and person_id (PK).

The next table created was that of ***label***. Label is very similar to people, except that it was a 1-M conversion and not a combination. All labels in the top100 table were taken and put into their own table. This table was never used but there is also a foreign key inside track to point to this table. This was chosen to be in a new table due to the fact that there are only a limited number of lables with several tracks pertaining to a label - and to conform to 3NF, this needs to be put into a new table so that the database can be normalized with reduced redundancies. Label has its own PK label_id.

After having completed label and people, ***track*** could be created as the two foreign keys it needs have been created. This table is the primary table, which consists of many different attributes - each will be discussed as to why they were included here and not in a separate table. First, the track was given its own track_id PK - instead of using the track_prefix. Although track_prefix could have been used (which is a combination of the track year and yearly position), a unique, unused key that is simple increment and standard numbering seemed cleaner. It's also important to note, because the prefix is unique, we could have used a compound key of the year and the yearly_rank in replacement of prefix. The following attributes were saved inside track, as their data is still informative: title, year, yearly rank, weeks charted, number of weeks in top 40/10, number of weeks peaked, highest rank, track time, track BPM (beats per minute), date entered, date peaked. With these attributes left, there are no transitive or partial dependencies left (besides that of the prefix, which was told to remain in the track table). Then, there are two foreign keys track_label_id and track_artist_id to connect the table to the two created above.

Now that track is complete, ***wrote*** table is made to link the m:n relationship between composers and track. This linking table is a little different from others in that is also includes the position of the composer from the original top100 table as a compound primary key with track_id.

And finally, the ***position*** table is implemented onto track using complex dating that simulates the repeating weeks group. The position table itself is composed of the most rows by far - and links to track with a foreign key. I did attempt to modularize the lengthy filling SQL code with a loop, but ran into a problem while incrementing week1, week2, week3… etc. I could combine the string week and an increment into a single string to produce “week1”, “week2”… but SQL itself has no way to evaluate a string as an attribute of a table. I resorted to the original SQL repeating code.

One of the only problems encountered was that of transferring dates between tables. When a date was null, it did not like that a date was ‘0000-00-00’ as a null date equates to - and threw an error. This was overcome by turning strict mode off inside of sql and restarting the server, which allows null date format.

Queries to the Database in SQL

The queries were rather simple to implement when one understood what needed to be in the “where” clause. First and foremost, anytime two tables were included where there was a foreign key, the keys had to be set equal with where. An example with people and artist: track_artist_id has to be compared to person_id so that only artists are grabbed from people.

As for the most complicated query, by far #8 was difficult with the aliases. After completing it, the logic makes sense, but while constructing the logic to put into SQL, it was very confusing - first, the only thing being printed are the results from one of the aliased track tables - the other track table was being compared, but it was never being printed. At first, I thought I would have to print both instances, but then later realized that this would created duplicates as they would be cross checked and printed both ways.

Conclusion

The database is in 3NF for the purpose of this project. All repeating groups have been dealt with, and all associative data put into new tables. There could be a relationships between a few of the attributes in track, but normalizing these more would create overhead to the database - and the level needed to normalize it completely is unnecessary. As it stands, the database is more efficient without this extreme normalization.

It’s also important to note that the database should delete the top100 table if it were to ever be finalized. Everything from top100 (that is needed) was taken from this in-between table. The database should be seen as the other tables and without that of top100.