COMP 3380 Winter 2023 Project

Part 3: Project Report

Group 16

Christie, Nikolaas
christ37@myumanitoba.ca

Duong, Edwin
duonge1@myumanitoba.ca

Zhang, Linpu
zhangl53@myumanitoba.ca

Summary of Our Data

The dataset that we chose for our group project was the Chinook database, which took real world data from Apple's iTunes, a digital media store. The database contains information pertaining to artists, albums, customers, employees, genres, invoices playlists and songs. Across 11 different .csv files the size totaled 611K. We have a total of 15,617 rows in our database. Originally, we had narrowed our options down to two databases, based on their quality and size, but ultimately chose the Chinook dataset, arbitrarily

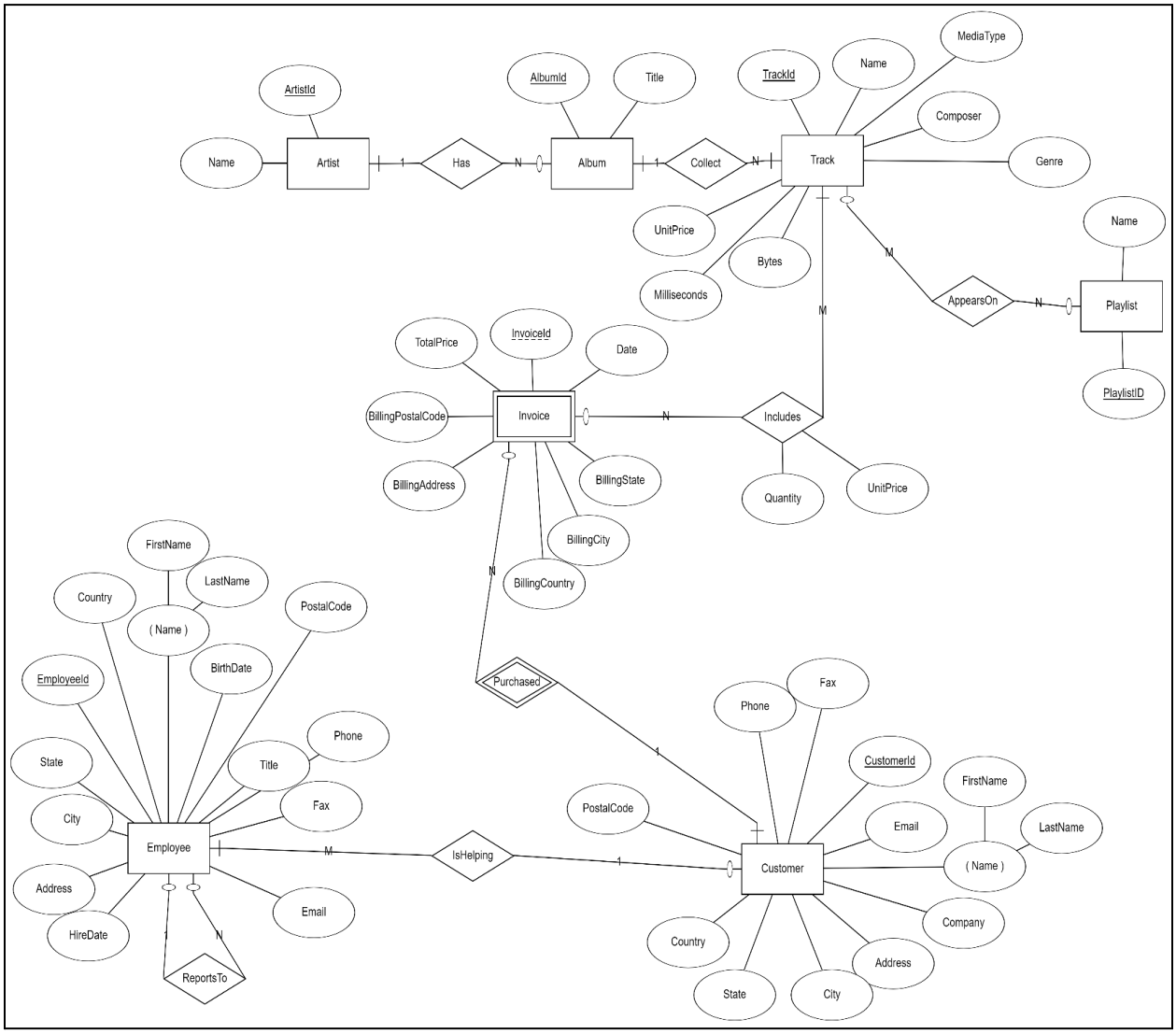

Figure 1 - EER Diagram

The data was broken down into 11 tables to reduce data duplication. It should be noted that there is some data duplication, notably the fact that an Album contains an ArtistID when it could be instead derived from the Tracks on the Album. One difficulty in modeling this data is the fact that Tracks and Albums can only have one artist. This poorly reflects the fact that many songs and even albums represent the work of more than one person. One way to fix this issue is to

create a separate table for Artists and the songs they sing, and, as well as their Albums. As we have seen in class, this is the only way we can model Many to Many relationships and would be required to fix the aforementioned issue.

Alternative Database Types

Despite the fact that a relational database was sufficient for this project, we felt that a graph database might have been a better fit considering the requirements of the data we were modeling. To create even the simplest interconnected queries, we needed to do 2-3 joins just to have the correct data. This could have been rectified by a Graph database because relationships like Artists, Albums and Songs are tightly interconnected and must be joined together frequently. Furthermore, the flexibility of dynamic typing affords us many advantages when modeling the music industry. Artists don't just release Albums but also Singles and Compilations. This would be hard to implement with a relational database. However, in a database we could just create a new type and connect the appropriate artists.

Another restriction, due to the fact that this database was implemented with a relational model is collaboration. Within this database both a song and an album can only have one artist. This doesn’t reflect the real world where many songs are the work of two or more artists. To represent this in a relational database we would need to create another table to represent this many-to-many relationship. It is clear that given that this database was relational impacted its flexibility in modeling real world relationships like the ones found in the music industry.

Problems We Encountered

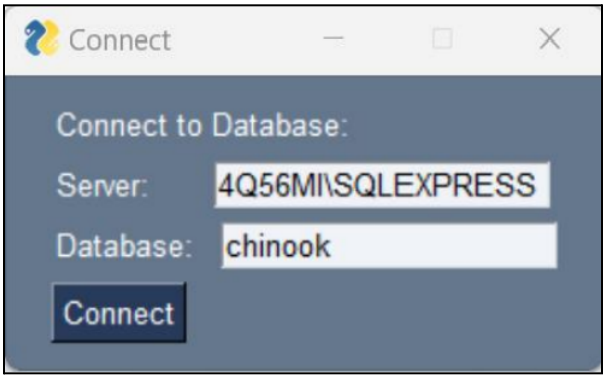

Fig 2.1 Connecting to the interface

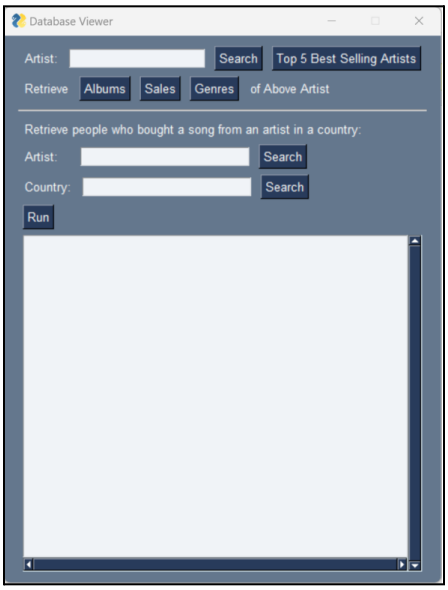

Fig 2.2 Main Interface Window

We originally planned on creating our interface in Microsoft Report Builder, however, we found the program difficult to set up. Instead we chose to create our interface in Python using a library named PySimpleGUI. We found PySimpleGUI to be a good fit for a project of this scope. It is fairly simple compared to alternative GUI frameworks like Windows Forms. Despite this we could easily see how a project like this could quickly balloon in complexity as the scope of the project increases. However, for a project of this scope it was beneficial to not have to deal with the bulk of a more industry focused library like Windows Forms and be able to focus on creating a prototype that was easy to program and just worked as an alternative.

One error we encountered when creating our interface was when connecting to our database, we set the timeout to 1. This worked on Leonard's machine but failed to connect on Nikolaas' machine. We solved this problem by removing the timeout to fix the bug. This taught us that it is important to test code on multiple platforms because a program working on your machine doesn't guarantee the same result on another's.

Interesting Queries

Some of the interesting queries that our program can execute are:

- Selecting the Top 5 best selling artists.
- Searching by any artists present in the database by their name.
- Getting all of the albums, sales figures, and genres that an artist makes.
- The names of all the customers who have purchased a single or album from an artist in a country.

We found these queries to be interesting because they leverage SQL to summarize data that isn't readily available. Each of the queries yield values that bridge at least three different relationships and give the user insight into important information that would be useful to a music industry insider.

Conclusion

We felt that this database would be a good example for teaching a class such as COMP 3380. We came to this conclusion because it would be a good exercise for future students in decomposing an EER diagram because it contains many different relationships. Furthermore, because of the size of the database, it could teach future students important lessons in query optimization because poorly crafted queries will take noticeably longer to execute.

Appendix

Source data was obtained from the Chinook sample database using the Microsoft Public License (Ms-PL).
https://github.com/w3c/csvw/tree/gh-pages/examples/tests/scenarios/chinook

EER Diagram image was made using ERDPlus.
https://erdplus.com/