Principles of Data Management: Phase 3 Report

Patrick Ehrenreich, Loren Davies, Michael Hopkins- The Disc Clones
Authored by Patrick Ehrenreich; Edited by Loren Davies and Michael Hopkins

Summary

For the purposes of our group project, we, the Disc Clones, chose to create an application centered on our appreciation for music, which we initially planned to model off the site discogs.com. The site itself is both an excellent model and data source; with a number of interesting features, it served as a major source of inspiration for our final product, as well as providing a significant chunk of our data (our samples being a meager subset of the site's massive collection, which exceeds 5GB in *compressed* form). We began with the simple goal of replicating the site's more basic functions, namely, efficient and well organized data storage, and a search function capable of querying any of the tables we established. As we progressed, however, our aspirations only grew, and with them the scope of the project. At the time of submission, we have a search function capable of querying each of the four main tables (Song, Album, Artist and Label), a robust web-based client capable not only of displaying the information contained within the database, but allowing the user to interact with and update the data in various forms, a fully-fledged administrative section for site managers to access, update and delete data, and even methods of event attendance tracking.

Phase Zero

Together, our group decided our project would be developing a http://discogs.com clone. Discogs allows you to input the music you own and store it on your account, views the collections of others, and sell parts of your own (physical) collection. Given the variety of data related to each individual artist or album, there is a significant range of possibilities present for creating a recommendation algorithm which could add depth to the project beyond simply creating a functional database. This would increase the complexity of the workload, but would certainly provide for some interesting results.

First a user will need to create an account, so everything they create can be associated with them. When a user would like to add music to their collection the application will ask for the artist/album, it will then check to see if the album is already present in the database. If the album is not present another screen will appear allowing the user to input more information so it can be added to the database. The user will also be able to go to their account and view their entire collection, the user would also be able to go to other user's accounts and view their collection. If a user decides that they want to put an item in their collection up for sale they can go to their collection and press a button to add it to the store. The user can pick what they would like to sell the album for and choose how long it can be in the store. Other users can then go into the store and browse records that other have put up for sale. The application may also be able to suggest the user other types of music based on what the user has in their collection.

Phase One

Our group has now decided for sure to create a music-oriented service modeled off that provided by Discogs itself. This will require a large amount of information, most of which can likely be retrieved directly from discogs itself. This information will include song names and metadata, artist names and metadata, album names and metadata as well as album editions, and label information. The database will also require a collection of users (which will have to be generated pseudo-data, since we can't just copy actual user information for privacy reasons). The users may be required to be a part of a separate database for security reasons. The two databases would then interact so that information regarding a user's personal collection could be retrieved (or, alternatively, all the data could be stored in a secure database). The database would have search capabilities as follows. Firstly, any individual quantity would be searchable without relying on other related characteristics (i.e. a user need not know anything about a label to find an artist or vice versa). Secondly, all users will have a publicly available collection which can be viewed by searching for the user's username (so long as the user does not manually choose to privatize their collection). Obviously, the database will include a full UI and be accessible via the web. The database should be useful for simply 'researching' a song/artist/album/etc beyond purchasing from a user's collection; it should have additional information such as lyrics or other interesting and relevant information to the items stored (think Spotify's Behind the Lyrics).

Phase Two

Normalization Analysis

As of this submission, our database is in third normal form. First Normal Form is satisfied by the fact that all data has been properly divided into database tables, and that these tables contain only atomic, non-repeating values. Second Normal Form is satisfied by the fact that all attributes in the current table-set serve to describe the object defined by the table's primary key, and that there are no non-key columns present that do not pertain to the primary key. We meet the requirements of Third Normal Form because with the inclusion of the foreign key dependency chain of Label -> Artist -> Album -> Song and the individual, secondary references included by table (Edition for album, for example), all transitive dependencies have been eliminated. Thus, the first three normal forms are satisfied, and the database is in Third Normal Form.

Progress Report

It has begun to appear as though our initial goal of replicating the Discogs service was, perhaps, a bit too ambitious. Though it would most certainly prove interesting and enjoyable to build a full, working replica of the service, the sheer complexity of interactions present in the Discogs service make taking on such a task impractical. However, we remain set on producing a product oriented toward music services. Though it may not have the robust predictive algorithms for music recommendation or buying/selling services, we still plan to create an engaging web application which allows users to perform actions such as rating their favorite tracks. We also still plan to have a fully functional search algorithm and interface implemented, and to include other features to whatever extent our availability and skill overlap.

Phase Three

As of the time of writing, our project has come further than we could've expected over the course of the previous months. After the borderline catastrophic rush caused by procrastination preceding Phase Two, we doubted that we'd get a chance to implement the majority of the features we originally wanted to include. However, we somehow found the time to power through not only the search function, but a fully functional user account system, an entire database administration system which allows for in-application manipulation of the data in the database, the inclusion of a fully-fledged rating system based entirely on user input, and the addition of event tracking functionality. Of course, this represents a fairly large deviation from our initially proposed structure for the project, and as such, we have included updated UML and ER diagrams in the submission file path "/src/docs/Phase 3". Below, I have included a list of major changes occurring during Phase 3:

- Deletion of the discography table (it provided no meaningful relationships which could not be found through other tables).
- Addition of given_ratings table (many-to-many table, used to track user ratings of specific tracks)
- Addition of events table and locations table (events table is responsible for tracking upcoming events related to artists tracked by our database; the locations table was added to ensure that the database remained in 3NF)
- Addition of event_attendees table (used to track user declarations of attendance in the context of those events present in the aforementioned events table)
- Refactoring of nearly all table attributes to better distinguish between similarly named attributes, as well as to enforce new non-null regulations and include better managed foreign keys
- Removal of the Artwork column from the Album and Edition tables, as it was essentially serving as dead weight in terms of storage and access times, and was not utilized in the web application.

Overall, our third phase of production was by far the most successful. Though it was (and still is) stressful and challenging, I feel as if the payoff has been largely in our favor.

Design Rationale

Though it was obviously not the best choice of language or framework available on the market, our group chose to work with Java and the Spark framework for the development of our web application. We are acutely aware of the drawbacks of the use of Java, though, in retrospect, still feel as if we made the correct choice of framework. Between the three group members, Java was the only common language, and given that the actual process of application development fell to me, I feel that Java was most certainly the best choice for our project; had I been forced to stumble through the process of designing our application in a foreign language and framework, I sincerely doubt that our results would have been as complete as they ended up being. Besides this, Spark also provides a fairly intuitive and simple environment for developing web applications in situations such as this; though the Spark/FreeMarker pairing does not function well in terms of scalability or dynamic design, it provides a great service in simplifying front-end development when back-end processes are the primary focus of the task at hand.

Improvements, Limitations, and Challenges

Despite the fact that our project progressed further than we truly expected, the development process was still fraught with a variety of challenges. For a start, our group had strongly conflicting schedules, which made meeting in person to collaborate on the project a righteous pain. The framework which the product was developed in also proved, in some cases, to be a hindrance. Despite my prior statement that the chosen Java/Spark framework was our best possible choice, I would be remiss to deny the limitations which the framework imposes. We would certainly have liked to have a more robust design scheme for the finished product; our final submission includes only basic CSS/HTML design, and lacks outside aesthetic injections such as those provided by React or Bootstrap which have the potential to greatly improve the visual appeal of a project. The reason for this is twofold. Firstly, the Java/Spark framework is not the easiest to integrate with outside libraries; secondly, we believed our development time was better spent focused on improving the features available to the user, instead of squandering it on a more aesthetic based focus. Regardless, if presented with future opportunity to rectify the project's shortcomings, we would be sure to improve the graphical validity of the application as a whole. Beyond this, we would also like to increase our focus on scalability. We currently utilize a PostGreSQL database with memory-based caching abilities. In the short term, this is an efficient model; however, were we to ever expand to a system as robust as our model site, Discogs, we would need to alter our practices to improve both scalability and efficiency. In the future, conversion to a multi-layered index model covering a significantly larger data set would be ideal for the purposes of a database with such a large and varied scope of functionality.