Movieverse!

G2 Software
dmc1542: Domenic Cacace
mjh9122: Michael Holtz
vfo1725: Victor Oliveira
sw1860: Seth Walker

February 18, 2022

1 Introduction

We at G2 software intend to develop a program within the movie domain as our primary choice. Our product aims to be a social platform that provides users an ecosystem centralized around the visual entertainment industry. We aim to simplify the movie-exploration and discussion experience, by providing the users with the ability to curate their own collections of movies, obtain recommendations based on their collections, and rank their favorites so that others can discover hidden gems or new blockbusters.

In the event that the movie domain is not available, then we intend to develop an application for the music domain as our backup option. In this case, our product would emulate much of the functionality present in the application described for the movie domain, however, instead of focusing on the film industry, we would prioritize recommendations and social connections centering around the music entertainment industry.

The backend of our program will be developed in the Java programming language, with a graphical user interface utilizing the JavaFX library for the front-end. The back-end of the service will implement the PostgreSQL database management system in which we will store information about films (or music), users, and other relevant information.

2 Design

2.1 Conceptual Model

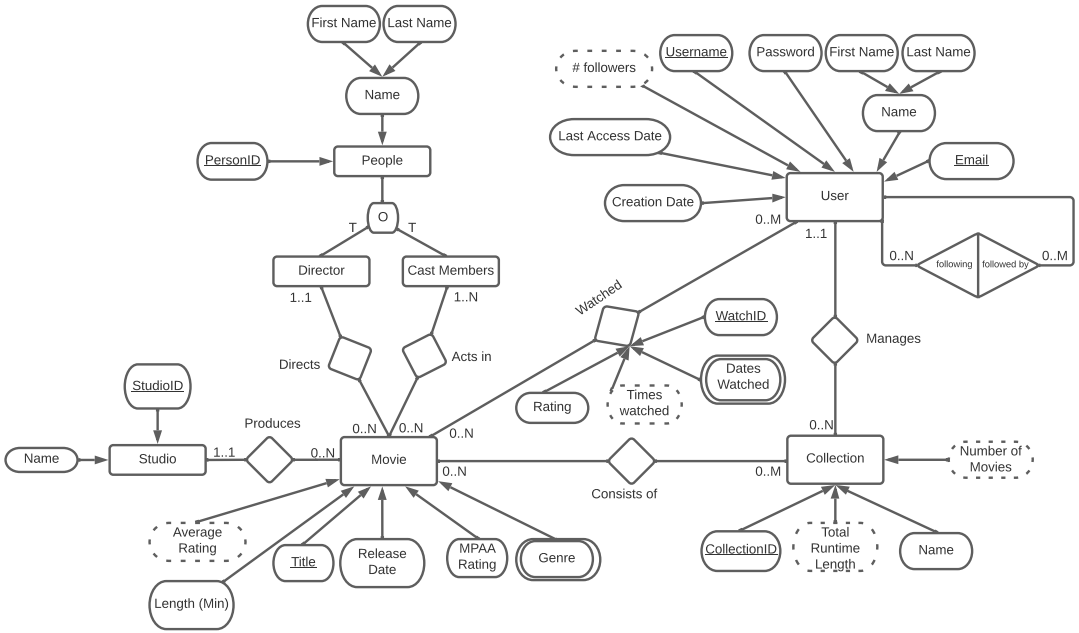

Figure 1: EER Diagram

Figure 1, seen above, is a conceptual data model of the database that will support the data and program requirements outlined in Section 1. In this section we provide a brief description of the entity, but primarily focus on the assumptions and design choices around the data model. Further descriptions about the attributes and entities can be found in Section 2.2: Reduction to tables, and data requirements can also be found in more detail in Section 2.3.

Users:

The User entity represents one client's account within the database. Two major relationships that User entities have to other entities include the Collection and Movie entities, where they track the user's collections and individual movies watched respectively. Some of our design choices include that only one user can ever create a collection, but that a user can create zero or multiple collections, which necessitates a 1:N cardinality, and is a partial participation. As it relates to the Movie table, the idea behind this relationship is that we can have any number of users watch any number of movies, so the relationship between these tables should naturally be a N:M cardinality. Note that this requires separate tables when translating to the logical model, but this is described more in Section 2.2. Lastly, we have also defined a friend to be someone whom a user is currently following, and being followed by, another user. However, this is distinct from followers, where any one person can follow another - but the mere action of following that user does not indicate friendship. Since we want to support any number of individual users following any number of other users, and be able to be followed by any number of users, we have another N:M cardinality to indicate this relationship.

Collections:

The Collections entity represents the playlists of movies that a given user can have stored in their account. A given user can have 0 or more collections stored at a time, thus being 1:N cardinality, and can manage their collections via creating and naming a new collection (with no movies in it), deleting an existing collection, and adding or deleting movies from any of their collections. There is also a N:M cardinality between movies and collections, because any one movie should not be limited to a singular collection, and any one collection should naturally not be limited to a singular movie.

Movies:
The Movies entity is the core of the product. It has a variety of relationships with other entities, many of which are described in their respective sections, but a brief description of the logic will be provided here. We are operating under the assumption that only a singular studio can produce a movie, but that a studio can produce zero or more movies. As a result, the relationship between Studio and Movies has a cardinality of 1:N. It should be noted that a movie must have a studio, so this is total participation. There is also a relationship between Movies and Directors, where we are similarly assuming that a singular director can direct a movie's production, but that a single director can direct zero or more movies. Using this assumption, we have established the cardinality to be a 1:N relationship. It should be noted that a movie must have a director, so this is total participation. Cast members are an exception to this trend; while they are also in the total participation category, we are assuming that there must be at least one cast member to make a movie, but there can be more than one as well. There are no restrictions on the number of movies on which a cast member can work, so they can be involved in anywhere from 0 to many movies inclusive. This creates a N:M cardinality between these entities. Lastly, with regards to the collections, there can be zero or more movies in a collection, and zero or more collections can have any one movie. Assuming that this non-distinct membership requirement is in place, this would create an N:M cardinality as well.

People:
The People entity is related to 0 or more movies, and it has specialization to the Director Entity and Cast_Member Entity. This specialization is overlapping and total as one person can be both a director and a cast member. The relationship between Director and Movies is 1:N (1..1 - 0..N) as every Movie needs exactly 1 director, while a director can direct 0 or more movies. The relationship between the Cast_Member and Movies is N:M (1..N - 0..M) as a Movie must have at least 1 actor who acts in it while a cast member can act in 0 or more movies. We are assuming that every person has a first and last name and their id will be handled incrementally behind the scenes.

Studio:
The Studio entity describes both the name and unique ID of a movie studio, with each studio in the database relating to the Movie entity via having each

studio produce 0..N movies in the database, thus being of 1:N cardinality. In this project, we assume that each studio must provide a name, and their id will be handled automatically by the database.

2.2 Reduction to tables

Collection(collectionID, *username* (foreign), Name)
Movie_Collection(*collectionID* (foreign), *title* (foreign))
User(Username, password, first_name, last_name, email, last_access, creation_date)
Following(*follower_username*(foreign), *followee_username*(foreign))
User_Movie(watchID, *username* (foreign), *title* (foreign), rating)
Dates_Watched(*watchID* (foreign), date)
Studio (StudioID, Name)
Movie(Title, length, release_date, MPAA_rating, *studio_id*(foreign), *director_id)*(foreign))
Movie_Cast_Member(*Title* (foreign), *personID* (foreign))
People(personID, first_name, last_name)
Director(*personID*(foreign))
Cast_Member(*personID* (foreign))
Genre(GenreID, *Title* (foreign))

User: The user entity's attributes are mostly directly related to the user and are not needed within any of the other entities, so these attributes are mostly singular. This includes the username, password, email, last access date, and creation date - which are all singular attributes that compose their own columns. The one exception is the name attribute, which is a composite attribute, which was translated to the table by making separate columns for the first name and last name attributes. There is one derived attribute, number of followers, which will be derived by counting all of the rows in the Following table where the followee_username is the current user's username.

Following: The Following table was created because of the N:M mapping with the "following/followed by" relationship, this relationship constitutes its own table where we can store two user's usernames in a row, where the first is the follower, and the second is the followee. This enables any one user to have multiple followers and follow multiple people without having redundant data in the User table.

Collection: This table represents a collection of movies that a user can build, and only has two simple attributes that are directly implemented as columns: collectionID and name. However, the information regarding what movie is in a collection is stored in another table because each table needs to support multiple movies - a requirement that we couldn't easily achieve by including movies within this Collection table. Therefore, this additional Movie_Collection table, described in more detail below, was made to accommodate each movie in a collection. There are two derived attributes associated with this table, one of which is total runtime - which can be derived by summing the runtime of each movie within the collection - and number of movies, which can be derived by counting the number of distinct titles within each collection.

Movie_Collection: This table records all the movies that are entered into a user's collection by holding the collectionID with which that row is associated, and the title of the movie. This decomposition was necessary because each collection needs to be able to store multiple movies, so by having this decomposition we can easily search for all movies in a collection by using the collectionID.

User_Movie: This table is a necessary decomposition because of the N:M relationship between the User and Movie entities. This table maintains a history of which user has watched which movie, and must include the username and title of the user and movie respectively in order to uniquely identify a tuple. Other attributes such as the times watched and rating from that user are also recorded so that we can support popularity gauges by customers, where times watched is derived and can be determined by counting the number of times a given username and movieID appear in the table.

Dates_Watched: This includes a watchID to uniquely identify the dates on which a user watched a particular movie, which is recorded in order to accurately gauge popularity of content. This is a separate table from the User_Movie because a movie can be watched repeatedly, and is therefore inherently multivalued. Using the watchID as a foreign key, we can uniquely record when a user watches a particular movie.

Studio: This table is necessary because of the 1:N relationship studios have with movies. By having a separate table it ensures that a single studio can

be responsible for many movies, but a movie will always have a studio. Furthermore a separate studio table ensures that each studio can be uniquely identified by an ID number and there are no anomalies when deleting movies - namely, deletion anomalies that would remove a studio from the database if the movie that a studio produced was deleted.

Movie: This table This entity records the details of a movie in the database. It has many simple attributes like title, length, release date, as well as the MPAA rating. Other attributes that identify the studio and director must also be included so that we can uniquely identify them, and also to prevent deletion anomalies that remove these entities from the database if the movie was ever removed. There is also an average rating attribute that can be derived by finding all of the ratings correlated with that movie title in the User_Movie table and dividing by the total number of ratings to yield an average.

Movie_Cast_Member: Since there is a N:M relationship between movies and cast members, we created a separate table to record which movies a cast member appears in by recording the movie title and the cast member's ID. In this way a cast member can work on multiple movies and a movie can have multiple cast members.

People: This table is one where it assigns a person an ID number and provides a first and last name. This table's purpose is to be a superclass for the specializations of Director and Cast Member, which have common attributes. The relationship is total and overlapping, meaning that a someone cannot be a "Person", they must be either a director or a cast member, or possibly both.

Director: This table is responsible for nothing more than identifying which Person objects are contained within the directors specialization for a movie, so that they can be entered into the movie entity

Cast_Member: Much like the director table, this table is merely a declarator for the Cast Member specialization. It identifies a person as a cast member so that they can be added to a movie entity.

Genre: The genre table is necessary because a movie can have more than one genre instead of simply one genre, so because of the multivalued nature

of the attribute we specified a new table where each row holds the movie Title, the GenreID, and the GenreName. This enables multiple genres to be stored for one movie, and accessible by its title.

2.3 Data Requirements/Constraints

Not Null Attributes:
Collection: Name
User: Password, first_name, last_name, email, last_access, creation_date
Dates_Watched: Date
Studio: Name
Movie: Length, Release_date, MPAA_rating, studio_id, director_id
People: first_name, last_name
Genre: Genre_name

2.4 Sample instance data

Collection Ex:
1.(1, *user1*, MyMovies)
2.(2, *user2*, SadMovies)
3.(3, *user3*, HappyMovies)
4.(4, *user4*, LongMovies)
5.(5, *user5*, ShortMovies)

Movie_Collection Ex:
1.(*1*,*2*)
2.(*2*,*4*)
3.(*3*,*6*)
4.(*4*,*8*)
5.(*5*,*10*)

User Ex:
1. (greg303, password1, Greg, Lee, greg@gmail.com, 2021-5-1, 2014-1-1)
2. (george204, password2, George, Martin, martin-is-amazing@gmail.com, 2020-1-5,2014-6-12)

3. (avery939, password3, Avery, Clark, avery939@gmail.com, 2021-7-5, 2014-5-7)
4. (coolkid503, password4, Carlos, Perez, cool-kid503@gmail.com, 2022-12-5, 2014-4-4)
5. (victor118, password5, Victor, Metic, victor118@gmail.com, 2021-10-23, 2016-6-6)

Following:
1. (*freddie1*, *maddie1*)
2. (*carson1*, *shannon1*)
3. (*ben1*, *emily1*)
4. (*peter1*, *tobias2*)
5. (*harry1*, *simon1*)

User_Movie Ex:
1.(1, *user1*, *Megamind*, 4)
2.(2, *user2*, *Jaws*, 4)
3.(3, *user3*, *Jaws 2*, 4)
4.(4, *user4*, *Finding Nemo*, 4)
5.(5, *user5*, *Finding Dory*, 4)

Dates_Watched Ex:
1. (*0213*, 2021-14-5)
2. (*0213*, 2021-3-12)
3. (*0213*, 2020-6-20)
4. (*0213*, 2021-3-4)
5. (*0213*, 2015-10-19)

Studio Ex:
1. (1, Dreamworks)
2. (2, Disney)
3. (3, Universal)
4. (4, Warner Bros)
5. (5, MGM)

Movie Ex:
1.(Megamind, 96, 2010-10-30, PG, *1,0006*)
2.(Lord of the Rings, 2001-12-19, *2*, *0007*)
3.(Finding Nemo, 120, 2003-05-30, *1*, *0008*)
4.(Jaws, 150, 1975-6-20, *3,0009*)
5.(Jurassic Park, 120, 1993-6-11, *4*, *0010*)

Movie_Cast_Member Ex:
1. (Star Wars, *0007*)
2. (Avengers, *0008*)
3. (Coco, *0003*)
4. (Ghostbusters, *0015*)
5. (Avatar, *0019*)

People Ex:
1. (0006, Tom, McGrath)
2. (0007, James, Cameron)
3. (0008, Steven, Spielberg)
4. (0009, Christopher, Nolan)
5. (0010, Peter, Jackson)

Director Ex:
1. (*0006*)
2. (*0007*)
3. (*0008*)
4. (*0009*)
5. (*0010*)

Cast_Member Ex:
1. (*0001*)
2. (*0002*)
3. (*0003*)

4. (*0004*)
5. (*0005*)

Genre Ex:
1. (1, *Husk*)
2. (2, *Lord of the Rings*)
3. (3, *Up*)
4. (4, *Spiderman*)
5. (5, *Titanic*)

3 Implementation

Use this section to describe the overall implementation of your database. Include samples of SQL statements to create the tables (DDL statements) and a description of the ETL process, including examples of the SQL insert statements used to populate each table initially.

Include also sample of the SQL insert statements used in your application program to insert new data in the database. Finally, add an appendix of all the SQL statements created in your application during Phase 4 and a description of the indexes created to boost the performance of your application.

4 Data Analysis

4.1 Hypothesis

Use this section to state the objectives of your data analysis; what are the observations you are expecting to find. Note that your final observations may end up differing from your proposal, that is also a valid result.

4.2 Data Preprocessing

Use this section to describe the preprocessing steps you have performed to prepare the data for the analytics. Preprocessing steps may include: data cleaning (e.g., filling missing values, fixing outliers), formatting the data (e.g., resolving issues like inconsistent abbreviations, multiples date format in the data), combining or splitting fields, add new information (data enrichment).

Explain how the data was extracted from the database for the analysis; if you used complex queries or views, or both.

4.3 Data Analytics & Visualization

Use this section to explain the process/techniques used to analyze the data, use data visualization to present the results, and explain them.

4.4 Conclusions

Use this section to explain the conclusions drawn from your data analysis.

5 Lessons Learned

In Phase 1 of the development of the project, we had originally planned on the creation of a chat feature that would allow users to chat with their friends while watching movies in order to enhance the moviegoing experience and increase the social interaction possible on the platform. However, by Phase 2 of the project, we had made the decision to completely remove this feature from the overall design while in the process of creating the EER diagram after deciding that the product delivered to the customer should primarily focus on the minimum functionality. It is more important to G2 Software to deliver a Minimum Viable Product (MVP) and prioritize our efforts on creating a bug-free experience.

The next subsection is meant to provide you with some help in dealing with figures, tables and references, as these are sometimes hard for folks new to LaTeX. Your figures and tables may be distributed all over your paper (not just here), as appropriate for your paper.

Table 1: Feelings about Issues

Flavor	Percentage	Comments
Issue 1	10%	Loved it a lot
Issue 2	20%	Disliked it immensely
Issue 3	30%	Didn't care one bit
Issue 4	40%	Duh?

Please delete the following subsection before you make any submissions!

5.1 Tables, Figures, and Citations/References

Tables, figures, and references in technical documents need to be presented correctly. As many students are not familiar with using these objects, here is a quick guide extracted from the ACM style guide.

First, note that figures in the report must be original, that is, created by the student: please do not cut-and-paste figures from any other paper or report you have read or website. Second, if you do need to include figures, they should be handled as demonstrated here. State that Figure **??** is a simple illustration used in the ACM Style sample document. Never refer to the figure below (or above) because figures may be placed by LaTeX at any appropriate location that can change when you recompile your source *.tex* file. Incidentally, in proper technical writing (for reasons beyond the scope of this discussion), table captions are above the table and figure captions are below the figure. So the truly junk information about flavors is shown in Table 1.

6 Resources

Include in this section the resources you have used in your project beyond the normal code development such as data sets or data analytic tools (i.e. Weka, R).