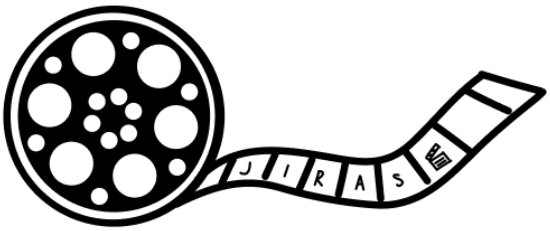
Database Design Document

JIRAS



Santoshni Birlangi
Akash Gajendra
Isaac Yeang
Joshua Johnson
Rong Xu

Executive Summary

The purpose of this database design is to assist two different types of users: content viewers and content analysts (of the streaming company) by creating useful recommendations and allowing for the analysis of attributes of the content viewer's movie viewing history. There are five main entities: analyst, user, movie, director, and actor. This is so the useful information for movie recommendations and for user movie viewing analysis is divided into several attributes organized by the entities, hence keeping everything simplistic. Some of the major attributes for the entities are analyst_ID, user_ID, movie_ID, name, genre, length, release_date, movie_description, actor_ID, director_ID, director_name, movies_starred, actor_name, actor_age, and actor_biography. The connections between analyst and user are zero to many, user and movie is many to many, movie and actor is many to many, and movie and the director is many to many. The primary interactions based on these connections are that the recommendations for users are based on actors and directors, in addition to that are analysts collecting data using users' viewing history and their watched movies list. The major assumption being made is that the target audience enjoyed their viewed movies, the watched list is based on movies that the audience has mostly watched and the users are interested in movies based on the actors or directors in them. The major risks are associated with using lists in our tables as it would increase search runtime by a linear factor and saving only three main actors for each movie (as it could lead to less viable recommendations).

Purpose

A content streaming company has a database of movies as well as statistics on their users and would like to make personalized movie recommendations. They have two types of users, content viewers and content analysts. The problem for content viewers is how to pick movies they would like to watch, and that is on the streaming platform, based on their collected information. The problem for the content analysts is allowing them to analyze the trends that are occurring among the content viewers.

The target audience for content viewers is people who watch movies. This ranges from younger kids whose parents put on movies for them, through adults who are watching movies to relax at night, to film students who are watching movies to learn the art of the craft. The reason this is the target audience is that the recommendation service should be applicable to anyone using the streaming service. And people who use the streaming service can be anyone who watches movies. It would not be helpful to aim a movie recommendation service at people who do not watch movies because there would not be much data to work from and they are unlikely to use the recommendations. The target audience for the content analysts is the streaming company. This is because they need to analyze the trends in movie watching in order to properly determine which movies can be dropped from the platform, as well as which kinds of movies, are very popular at the moment to try and get more like them on the platform.

The proposed solution is that we track the movies that people watch, along with the director and actors in those movies. The recommendation system will work primarily on two pieces of information, the director and the actors. One method that will be used to create recommendations is by finding other movies that were created by the directors of recently watched movies. Another method that will be used is by finding movies that have a similar cast as the movies that the viewer watched recently.

High-Level Entity Design

1. Analyst

   Purpose:

   The purpose of this entity is to distinguish between different analysts. We suppose this project is used for a large streaming company that includes a bunch of analysts assigned to analyze different users. In this case, the entity of analysts will track each analyst and the users they analyze. The Primary Key (PK) for this entity is [int] analyst_ID.

   Justification:

Besides the reasons above, we assume in order to know the current video streaming trends, analysts will have to track multiple users' watching history. That's why we also need an entity for tracking those analysts.

System Role:

The role this entity will play in the system is to keep track of every individual analyst, with attributes like an ID, and their analyzed user ID's and thereby, the trends they notice.

2. User

Purpose:

The purpose of this entity is to organize the users' watching history. We assume we have a bunch of users that need to be analyzed, and for each user, we also have their watching history of different movies which are also recorded in our database. The PK for this entity is [int] user_ID.

Justification:

We have this entity to recommend movies based on what they watched before (more acceptable and reasonable). On the other hand, this entity also helps analysts to define the trending content based on the users who have similar watching histories.

System Role:

The role this entity will play in the system is to track every viewer/user's movie choices, so the recommendations based on their viewing history are beneficial to the user; the attributes would primarily be the movies' IDs.

3. Movie

Purpose:

The purpose of this entity is to store all the related information about different movies. It's the main part of our database. This entity stored all of the movies either users watched or not. Since the purpose of this project is to create a recommendation system based on video streaming trends. This entity makes the main "connection" of other entities. The PK for this entity is [int] movie_ID.

Justification:

We have this entity to organize different movies' information. In this case, we assume we have a bunch of movies. Based on the users' watching history, we are able to recommend movies to them. This entity is necessary and important for data mining. The entity is closely related to the user, director, and actor entities.

System Role:

The role this entity will play in the system is to contain important attributes like the title, genre and will be tied to the director, actor, as some users prefer movies of specific directors and actors and viewing recommendations by genre, and their interest can be piqued by the title.

4. Director

Purpose:

The purpose of this entity is to store the information of different directors. It's one of the main entities we will make the recommendation decision. In this entity, it stores information such as director name, director works, and introduction for each director we have based on the movies we recorded. The PK for this entity is [int] director_ID.

Justification:

We have this entity to organize the directors we have. We assume to recommend users movies based on the movies they watch. To be more specific, based on the director who directed those movies, we will recommend related movies of this director. This entity is closely related to the movie entity.

System Role:

The role this entity will play in the system is to use user viewing history to create recommendations based on the directors of the movies they watch in conjunction with the movies ID attribute, basing the recommendations partially on genre and actors IDs as well.

5. Actor

Purpose:

The purpose of this entity is to store the information of different actors. It's one of the main entities we will make the recommendation decision. In this entity, it stores information such as name, works, and introduction for each actor we have based on the movies we recorded. The PK for this entity is [int] actor_ID.

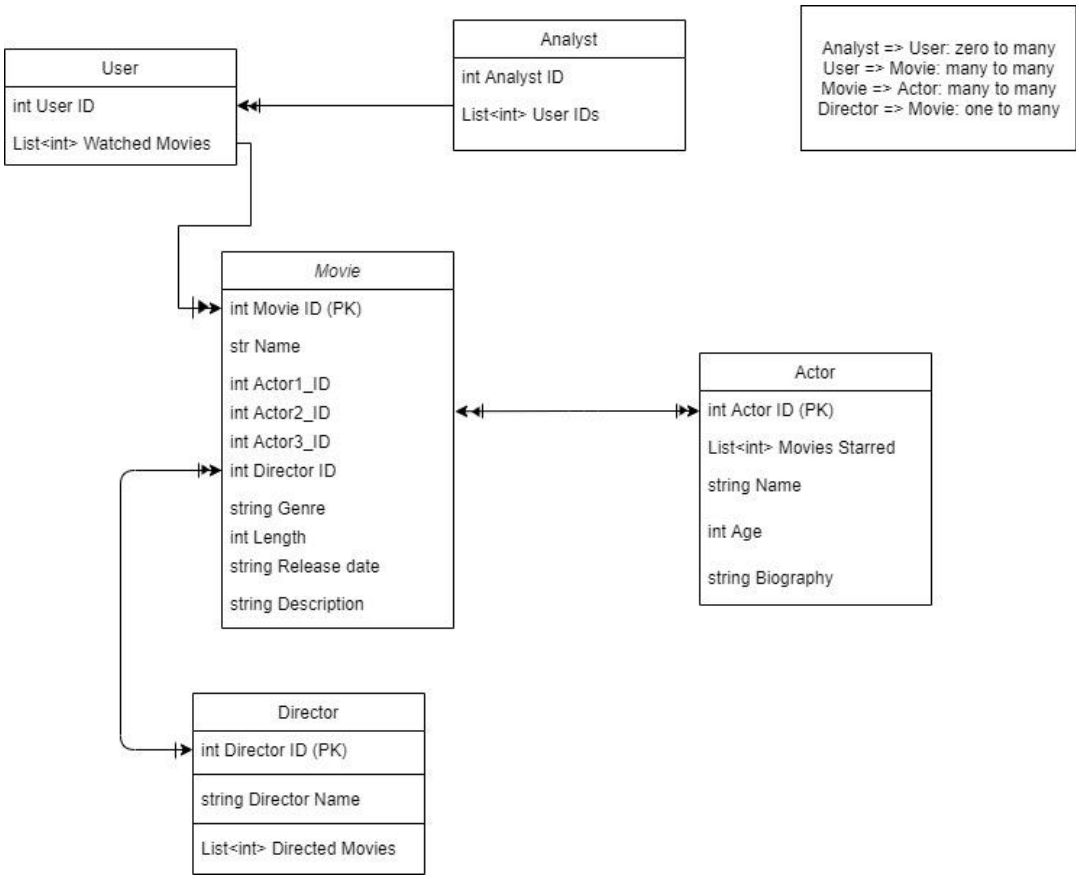
Justification:

We have this entity to organize the actors we have. We assume to recommend users movies based on the movies they watch. To be more specific, based on the actors who acted in those movies, we will recommend related movies of these actors. This entity is closely related to the movie entity.

System Role:

The role this entity will play in the system is to use user viewing history to create recommendations based on the actors in the movies they watch in conjunction with the movies ID attribute, basing the recommendations partially on genre and directors IDs as well.

Low-Level Entity Design



As showcased in the above UML diagram, our database is composed of five separate entities - Analyst, User, Movie, Actor, and Director - with their own distinct relationships. Each of these entities plays an important role in the big picture design of our client's required database. The purpose of the Analyst entity is to keep track of who has permission to read user data (ie: User ID, Watched Movies). In our scenario, this would be the database that the company uses to keep track of which workers have access to sensitive information and to utilize it to track streaming trends. The User entity keeps track of User IDs the corresponding list of watched movies (in the form of Movie IDs). These movie IDs correspond to the Movie entity which keeps track of all movies within the database and stores vital information about them. For example, the Movie entity stores three actor IDs (considering runtime), who played large roles in the movie (for other movie recommendation purposes). These actor IDs correspond to the Actor entity which stores vital information as well as a list of movies that the actor has appeared in. The Movie entity also stores the director's ID, which

then corresponds to the Director entity, allowing the Analyst to recommend other movies the director has worked on.

Entity Attributes

Entities:

1. **Analyst** - This entity serves to provide analytics surrounding the content streaming trend for content analysts. Each analyst is assigned an Analyst ID along with a list of User IDs to view any user-specific trends - In a nutshell, an analyst has read-only access to every aspect of the database design. As discussed in the previous section, the analyst ID serves as a credential and allows our client to track who has accessed sensitive user data.
2. **User** - The User entity consists of a User ID and a list of Movie IDs corresponding to the movies that the specific user has watched. One could make an argument that we could have added movie-specific details within this table, however, our team made a conscious design choice to avoid creating multiple entries for each user within this table and create another movie table that ties back to the User entity through the list of Movie IDs.
3. **Movie** - The Movie entity was created to be the one-stop for all movie-related information. It uses Movie ID as a primary key while hosting data in attributes like Movie Name, Genre, Description, etc. Instead of crowding this particular table with actor and director-specific details, we decided to assign Actor ID and Director ID attributes that tie to another entity for each.
4. **Director** - This entity was created to store director information and a list of movie IDs associated with other movies that they have directed. The table has a primary key in the form of a Director ID, additional attributes in Director Name, and Directed Movies. Our team intends to issue content-based recommendations using Directed Movies as it's likely that a user that likes a particular director's movie might like other movies by the same director.
5. **Actor** - The Actor entity ties back to the Movie entity via three attributes - Actor1_ID, Actor2_ID, Actor3_ID. This consists of actor-specific attributes like Actor Name, Age, and Bio along with Starred Movies i.e a list of movies within which they have been a part of the cast. Initially, we intended to make a Crew/Cast entity with a list of Actor IDs that tie to a slightly modified version of the Actor table. However, upon further reflection, the team decided to scrap the idea and instead implement three main Actor ID attributes within the Movie table that tie back to the Actor entity. Our approach with this database design has always been one that views unnecessary complexity as a poor design choice.

Entity Relationships

The entity relationships within the database are as follows:

- Analyst <=> User : Zero to Many
  Connection Attribute: User ID (Analyst -> User)

From the user's perspective, there is no quantifiable connection to the analyst. However, the analyst has access to multiple User IDs which makes this a **zero-to-many** relationship.

- User <=> Movie: Many to Many
  Connection Attribute: Movie ID (User <-> Movie)
  Each user has a connection to multiple movies through the Movie IDs and each Movie ID can be a part of the Watched Movies attribute for multiple users. Hence, this is a **many-to-many** relationship.
- Movie <=> Actor: Many to Many
  Connection Attributes: Actor ID (Movie -> Actor), Movie ID (Actor -> Movie)
  Each movie within the Movie entity has 3 Actor IDs embedded within it as attributes that are related to the Actor table. That being said, each actor within the Actor table can be a part of more than one movie. This dynamic makes this a many-to-many relationship.
- Movie <=> Director: Many to One
  Connection Attributes: Director ID (Movie -> Director), Movie ID (Director -> Movie)
  There are occurrences with duo-directors but our team for the most part has assumed that any given movie would have only one director. Any entry on the Movie entity has only one director but any entry on the Director entity ties back to multiple entries on the Movie table through the Directed Movies attribute.

Interactions

One major feature of our system is the recommendation of movies that occurs between the Actor and User entities. Based on the user's recently watched movies (list of movie IDs), we check the Movie entity (query from movie ID) for the three primary actors (stored within Movie entity's columns) within that movie and recommend the user other movies those actors worked on (stored in Actor entity).

Additionally, we have an interaction between the Director and User entities in which we check user's recently watched movies and check the corresponding Movie entity. Based on the director that led the movie, we recommend the user to other movies that the director has worked on.

The Analyst entity was primarily created to fulfill the client's requirement to run big data analytics surrounding user trends on their platform. The analyst by virtue of their read-only access to the entire database can run queries using the **User IDs** and their **Watched Movies attributes** to gauge trending/popular movies of the day. They could also observe large-scale trends in terms of average watch-time, regional distribution, demographic distribution, etc. which would help tailor the platform to their target audiences.

Assumptions and Risks

Within our proposed solution, we are assuming that the target audience enjoyed the movies they watched. Additionally, we are assuming that the company that uses our technology only adds the movie to the user's watched list if the user makes it through a certain percentage of the movie. For example, a poor implementation would be to add the movie as soon as the audience presses play. In such a scenario, the audience might not necessarily like the movie and shut it off immediately. However, if the movie was added to the user's watched list, then recommendations might become skewed and therefore unrepresentative of the movies the user actually liked. Another assumption is that the user will be interested in movies based on the actors that starred in the movie most recently watched.

With our proposed solution, we run the risk of increasing search runtime by a linear factor when using lists in any of the tables. For example, for the movie junkies that watch enormous amounts of movies, their movie watch list expanding may cause a database slowdown server-side. Another tradeoff we risk running would be only listing three main actors for each movie. Although this allows us to get the actors in a movie in constant time, this comes with the tradeoff of greatly reducing the number of actors we store (which we decided was worth it in our design process, nonetheless important to note).