Relational Schema:

```
Users(
  UserID: INT [PK],
  Username: VARCHAR(50),
  Password: VARCHAR(255),  -- Hashed
  Email: VARCHAR(255)
)
```
Login Information for Users, Could be sensitive and might need access control

```
Actors(
  ActorID: INT [PK],
  Name: VARCHAR(100),
  BirthYear: YEAR,
  Country: VARCHAR(50),
)
```
Actor Information from IMDB

```
Movies(
  MovieID: INT [PK],
  DirectorID: INT [FK to Directors.DirectorID],
  Rating: INT, -- (1-10),
  Genre: VARCHAR(50),
  Title: VARCHAR(100),
  Year: YEAR,
  Region: VARCHAR(50),
)
```
Movie Information from IMDB

```
WatchHistory(
  UserID: INT [FK to Users.UserID],
  MovieID: INT [FK to Movies.MovieID],
  WatchedOn: DATETIME,
  UserRating: INT -- (1-10)
)
```
User Watch History and Preference on a movie, contains privacy and should be only restricted to be accessible from users and their friends.

```
Genres(
  GenreID: INT [PK],
  Name: VARCHAR(50),
  Description: VARCHAR(255)
)
```
Genre Information from IMDB

```
GenrePreferences(
  UserID: INT [FK to Users.UserID],
  GenreID: INT [FK to Genres.GenreID],
  Rating: INT -- User's preference ranking for the genre(1-10)
)
```
User Preference on a genre, contains privacy and should be only restricted to be accessible from users and their friends.

```
ActorPreferences(
  UserID: INT [FK to Users.UserID],
  ActorID: INT [FK to Actors.ActorID],
  Ranking: INT -- User's preference ranking for the actor (1-10)
)
```
User Preference on an actor , contains privacy and should be only restricted to be accessible from users and their friends.

```
Directors(
  DirectorID: INT [PK],
  Name: VARCHAR(100),
  BirthYear: YEAR,
  Country: VARCHAR(50)
)
```
Director Information from IMDB

```
DirectorPreferences(
  UserID: INT [FK to Users.UserID],
   DirectorID: INT [FK to Directors.DirectorID],
  Ranking: INT -- User's preference ranking for the director (1-10)
)
```

User Preference on a director, contains privacy and should be only restricted to be accessible from users and their friends.

```
Friends(
  UserID1: INT [FK to Users.UserID],
  UserID2: INT [FK to Users.UserID]
)
```
Friendship relation is bidirectional and could be deleted from either side, can only be added when both agreed.

```
MovieActors(
  MovieID: INT [FK to Movies.MovieID],
  ActorID: INT [FK to Actors.ActorID],
  Role: VARCHAR(50)
)
```
Movie actor list Information from IMDB

Explanation of Entities and Relationships:

Entities:

- Users is an entity because it represents the virtual account that a person creates with the unique login information that they provide, which becomes the attributes of the Users entity.
- Actors is an entity because it represents real-world actors and can only be described by other attributes.
- Movies is an entity because it represents real-world movies that were produced and thus can only be described by other attributes.
- Directors is an entity because it represents real-world directors and can only be described by other attributes.
- Genres is an entity because it represents all the possible genres for the movies.

Relationships:

- “Friends” is a same-entity relationship set with the Users entity. It’s cardinality is many-to-many because any number of users should be able to be friends with any number of other users.
- “WatchHistory” is a relationship set from the Users entity to the Movies entity. Whenever a user watches a movie, a “watched movie” entry is created. The cardinality is many-to-many because a user can have any number of movies in their watch history, and a movie can be watched by any number of users.
- “GenrePreferences” is a relationship set from the Users entity to the Genres entity with a cardinality of many-to-many. A user will rate all the available genres, which will form their genre preference list, and a genre can be rated by any number of users.
- “ActorPreferences” is a relationship set from the Users entity to the Actors entity with a cardinality of many-to-many. A user will rate all the actors they have seen, which will form their actor preference list. An actor can be rated by any number of users
- “DirectorPreferences” is a relationship set from the Users entity to the Directors entity with a cardinality of many-to-many. A user will rate all the directors responsible for the movies they have seen, and a director can be rated by any number of users.
- “Movie Actors” is a relationship set between the Actors and Movies entities with a cardinality of many-to-many. Movies cast multiple actors and actors can star in roles in multiple movies.
- “Produced” is a relationship from Directors to Movies with a cardinality of one-to-many. A Director can produce any number of movies but a movie can only have one director.
- “Has” is a relationship from Genres to Movies with a cardinality of many-to-many because multiple movies can fall under one genre category and a movie can have multiple genres that it belongs to.

Design explanation:

Dataset From IMDB will be formed into tables of Movies,Directors,Actors,MovieActors. This design choice can better describe relationships between users and movies. A user might be interested in a particular movie, or they could be a fan of a director or an actor.

A generated dataset of a social network between users will be implemented for testing purposes.

Most of our entities user, friend, ActorPreferences, DirectorPreferences, GenrePreferences and WatchHistory are used to describe user preferences so that we can have enough information for our recommendation system. These relationships can be inserted, updated, or deleted frequently as our recommendation system interacts with users, so it is necessary to keep them in separate tables. For example, we might not want to update the user or actor table when a user hits the like button to an actor.

Recommendation system:

To fully understand the design, one needs to understand the recommendation system it was built around.

We have multiple ways to implement a Recommendation system.
When a user first registers the website. The user will select roughly 3 actors, directors, and genres to rate for movie recommendations. The user can then access the friends system to find familiar people and get recommendations based on the friends' watch history and ratings.

When a user generated enough relations with the system, a scoring based recommendation system will recommend movies:

Pseudocode:

```
    TotalScore = Movies( Rating: INT, -- (1-10)) +
    {FOR EACH FRIEND}: WatchHistory(UserRating: INT -- (1-10)) where MovieID==
    Movies.MovieID +
    GenrePreferences(Ranking: INT -- User's preference ranking for the genre (1-10))
    where MovieID== Movies.MovieID +
    {FOR EACH ACTOR}: ActorPreferences(Ranking: INT -- User's preference ranking for
    the actor (1-10)) where MovieID== Movies.MovieID +
    DirectorPreferences(Ranking: INT -- User's preference ranking for the director (1-10))
    where MovieID== Movies.MovieID
```

Digestible explanation:

Sum a core based on user's preferences and friends' ratings of movies.

Exclude movies from user's WatchHistory

Then get the Top 10-15 movies for user to choose from

We can also run a machine learning model periodically and recommend new friends and new movies.

Normalization:
For normalization,we want BCNF because it makes the schema more robust, by getting rid of implicit dependencies. Decomposing the entities further to explicitly state dependencies is beneficial to us because our recommendation system will have trouble joining tables otherwise. The stricter norm will make a more complex schema now but reduce dependency issues further down the line. For each functional dependency the left-hand side is a superkey.
There is no functional dependency where a non-superkey determines another attribute.