Student Number

Teammate 1: 202004055

Teammate 2: 201903318

Candidate Number

Teammate 1: 34382

Teammate 2: 36540

Contents

1 Application and Data

1.1 Application

We aim to design a real-life website application database that allows users to create a login account, give reviews, rate and comment on various movies and tv shows. We then want to leverage the user reviews to rank shows accordingly and show the result on the website. The website also contains relevant information regarding thousands of movies and tv shows for users to digest so that they can provide high-quality reviews while interacting with other users. Information includes but is not limited to titles, actors, release year, etc. We hope to create many relations and complex queries from this interesting application.

1.2 ER Model of the Application

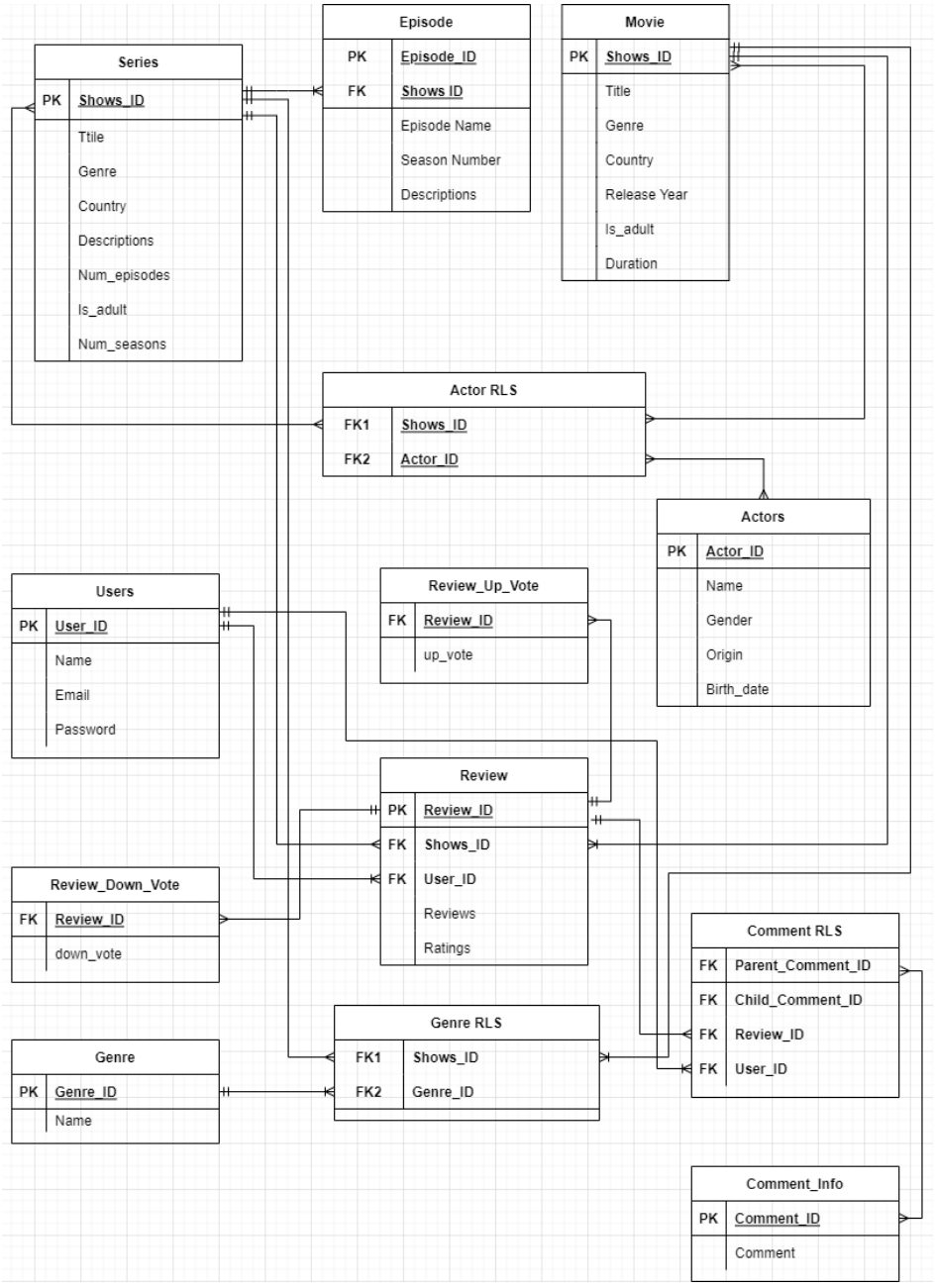

Figure 1: ER-Model of the Website application

1.2.1 Entities and Relationships

Figure 1 summarises the relationships and entities of the database application. Below is a detailed explanation of each table and their relationships:

Movie : This table stores relevant information about each movie. Each movie will have multiple reviews written by multiple users and each of the reviews will contain comments.

Series : This table keeps relevant information about each series. Each series will have multiple episodes, and each series will contain multiple reviews just like the movie table.

Genre : This table contains a list of all available show genres.

Genre RLS : Since each show can be categorised into multiple genres, this table is responsible for storing the relationships of the multiple genres associated for each show.

Episode : This table contains all tv show episode information. The relationship of the episode and series is many to one. Unfortunately, we are unable to find data for this part. Hence, the database creation ignores this entity entirely.

Actor : This table keeps all relevant information about actors. Each actor can act in multiple series and movies.

Actor RLS :This table is responsible for linking many to many relationships between the actor table to the series and movie tables.

Review : This table stores reviews of movies and tv shows written by users. As mentioned each of the movies and series will contain multiple reviews and the relationship between these tables is one to many (reviews).

Comment_info : This table stores all comments written by website users. Each comment is associated with the user, the review and any sub comments.

Comment_RLS : This table helps the database to connect various relationships that exist between users, review and comment entities. Each of the reviews has multiple comments from many users. Also, each comment can have multiple child comments . In other words, users are allowed to sub comment on another comment. All the relationships are shown in the ER model.

User : This table stores the login details and personal information of a user.

1.2.2 Normalisation

In our application, we use normalisation techniques to organise data efficiently by eliminating data redundancy and multiple values in attributes. Comment RLS, Table RLS and Genre RLS

tables containing the relationships between entities involved are the result of normalisation on Movie, Series and Review tables.

This technique is to reduce redundancy in a table. Not only this, it allows efficient streamline of operations like insert, update, delete and others. Lastly, normalising also enables us to save storage space. Minimising storage cost is especially important in the actual application when the database carries billions of data.

1.3 Data

1.3.1 Data Source

We are using data from multiple sources to fabricate data that we need. Below is the link to our original source of data (without any manipulation):

1. Netflix Movies and TV Shows | Kaggle
2. IMDB Dataset of 50K Movie Reviews | Kaggle
3. Online Data Generator
4. Sarcastic Comments - REDDIT | Kaggle

1.3.2 Data Manipulation

The data that we gather online is not perfectly aligned with our application data requirement. Hence, we decided to do data manipulation on our own using python to generate synthetic data that would then be used to populate our database. All the data we have in the database do not reflect the actual world, their aim is to ensure that we can test our database to work as intended. This is the most tedious and time-consuming part of the project.

Data generation and manipulation are done in two notebooks. One can find both process for each table as follows

1. *"/Answer Folder/Data Cleaning 1.ipynb" (Path)*
    - Movie and Series
    - Comment_Info
    - RLS_Comment
    - Genre
    - RLS_genre
2. *"/Answer Folder/Data Cleaning 2.ipynb" (Path)*
    - Movie and Series
    - RLS actor
    - Review

The generated data we used to populate the database can be accessed here *"/database_data/populate_data" (Path).* The original data file is for internal purposes only, one may ignore this.

2 Technology

2.1 SQLite: Relational Database

Through discussion within our group, we decided to use a relational database. A relational database will bring tremendous benefits to the application that we are building.

2.1.1 Costs and Benefits

Benefits

Our application will be considered a high complexity model. Given that the nature of social data has various relations, to users in particular, using a relational database will preserve the relational aspect of the whole database. This is because the relational database model puts data into tables and contains keys to relate to other tables. Unlike a no-relation database where it is object-oriented and will cause more confusion as the relationship complexity increases.

In addition, the API usage of our application will be high. This also means that retrieving information rate is higher as well. Using a relational database will greatly increase the efficiency of querying data from the database.

Lastly, our application also contains a lot of information. Hence, using a structured model will be able to prevent leakage of data, preserving the data integrity of our application. This allows the application to have a lower chance of running into error 404 (data not found).

Costs

As we look deeper, our application will consist of millions of rows of data (referring to the real IMDb). Hence, using a relational database might cause some data loss due to the increasing amount of tables in the database. However, we can solve these issues by carefully planning our tables of data and optimising by only creating necessary tables.

2.1.2 Alternatives

Graph Database

For future improvements of the project, graph databases can possibly be employed. It is a powerful tool when it comes to handling interconnected data which is the case for our website application. We can greatly leverage the speed-optimised query performance for data retrieval processes carried out in graph databases especially when the amount of data is tremendously large and is constantly evolving. However, we decided to proceed with a relational database instead for this project because of a few reasons below.

1. The database constraints for free users limit the number of nodes that can be created. In this project, we have more than a million records in total to deal with.
2. Graph data model is fairly new and we as a team are less familiar with the technical side of the graph database.

3 Operations

Here we will talk about how our DDL and other operations are applicable to the database application that we are building. The database creation code can be accessed here : */Answer Folder/Data Modelling and Creation.ipynb" (Path)*

3.1 Database Creation

3.1.1 Views

Three views are created to supplement the website application.

1. **Movie Ratings**
   Movie ratings view is created to join review, upvote, downvote and movie tables into one table that contains aggregated data for each movie.
2. **TV Show Ratings**
   TV show ratings view is created to join review and TV show tables into one table that contains aggregated data for each TV show.
3. **Review Ratings**
   Review ratings view is created to join comment rls, upvote, downvote and review tables into one table that summarises aggregated data for each review.

These three tables involve complex aggregation and multiple table joins that may result in complicated queries if to be used in a single query. Hence, creating a virtual table, not stored permanently can simplify the SQL code. We will talk about the view application in the queries section.

3.1.2 Triggers

A few triggers are created in the database to ensure any update and insert operations are carried out according to database requirements and website applications.

1. Email Verification: A trigger ensures that for every new account created, the email address given by the users is an actual email address. For existing users, this email address requirement is also checked when they are updating their emails.
2. Account Creation: An account log table is created to keep track of new accounts created and account details updates. This store information can be used for internal purposes such as website traffic analysis. A trigger is also set to guarantee that no duplicate email is used twice to create another new account.
3. New Shows: when a new show is added into the database, a trigger makes sure that the data about the new show duration and the release year are consistent. Duration has to be in seasons for TV shows while in minutes for Movies. A new show with a future release year is not allowed to be added to the database since nobody can give a review of something they have yet to watch.
4. Review Ratings: a trigger keeps the rating value to be within a range of 0 and 10.
5. Comment depth control: Two-level hierarchy comment, consisting of a parent comment and child comments, is employed. To enforce this, A sub (child) comment is given a

special ID that is different from its parent. A trigger makes sure that an insert transaction will be deported and flagged if a child ID type is to be included in a parent column.

3.1.3 Constraints

Website application database constraints are used to safeguard the accuracy and reliability of data in the table. Any violations occur, the transaction is aborted. Most of the constraints are programmed into the database when the table for each entity is created. The constraints include

1. Each movie, tv show, user, review, comment, actor and genre has a unique ID as a primary key so that they can be uniquely identified.
2. For a review and comment to exist, an associated user account and relevant show must exist as well. This is made possible through foreign keys.
3. When a user account is deleted, all information from other entities related to the user will be deleted as well that includes review, comment, rating etc. This is done through specifying cascade delete on the foreign-primary-key relationships in each table. The same concept is applied when a movie or tv show is deleted as well.

3.1.4 Indexes

Index is created on a title attribute both in series and movie tables to quicken the search engine of our website to search for a given show title that a user types in. As the number of users expands, search operation for a given show occurs more frequently than insert operation of the movie and series table. Hence, the benefit of having this speedy index may exceed the cost of insertion.

3.1.5 Queries/Updates

Multiple queries are run through the database to retrieve data needed for the website. They are listed below with explanations.

1. **Top 100 rated movies and tv shows**
   Justification : This table will be used as an interface of the website to let users know the most rated movies and tv shows on the website.
   Tables involved : Movie, Series, Review
   Mechanics : Union operation is used to merge two views, movie_ratings and show_ratings, vertically. The resulting table is then ordered decreasing by ratings and only the first 100 rows are displayed on the table.
   Outputs: Title, Average Ratings, Type

2. **Worst rated british movies**

   Justification : Movie ranking table that lets users know the bottom 10 british movies of all time

   Tables involved : Movie, Review

   Mechanics : Movie table is filtered on a condition that the movie country is the United Kingdom. The resulting British movie table is joined with a review table through show_id. The output table is ordered ascendingly on average ratings that have been aggregated for each movie. Lastly, rows are limited to only the top 10 for display.

   Outputs : Title, Average Ratings

3. **Top Review**

   Justification : The review rank is used so that website users can interact with the most famous reviews on the website. It also gives a signal to the users on which review is to be trusted.

   Tables involved : Up Vote, Down Vote, Review, Comment RLS, Movie, Series, Users

   Mechanics : Reviews by users are ranked based on engagement score. Engagement score is an average of vote score and number of comments received for each review. Vote score is calculated by subtracting downvotes from upvotes. The review_rating view joins upvote, downvote, comment rls and review table together and aggregates the scores. The view is linked to users through user_id, and to series and movie tables through show_id. The resulting table is ordered decreasingly based on engagement score

   Outputs : User Name, Reviews, Title, Type, Review Votes, Number of Comments, Engagement Score

4. **Movies with the most review upvotes and downvotes.**

   Justification : For this query, it is very applicable in our application because it provides an algorithm to suggest movies based on review upvotes and downvotes. Hence, allowing the application to suggest movies or series to users based on reviews.

   Tables involved : Up Vote, Down Vote, Review, Comment RLS, Movie

   Mechanics : A movie rating view is used here to simplify the query. The view is arranged decreasingly on review score for each movie. Review score is calculated the same way as vote score but aggregated based on movie show id.

   Outputs : Title, Review Score

5. **Movies with the most comment engagement**

   Justification :To keep users hooked on the website, this query enables users to participate in discussions about trending movies and series where many users are hype about. Having this allows application engineers to use such information to build a suggestion algorithm for the user.

   Tables involved : Review, Comment RLS, Movie

   Mechanics : Each movie from the movie table is joined to the review table on show_id key. The resulting table is then joined to comment RLS on review_id. The output table is then grouped by a movie so that the total comments (engagement score) can be summed for each movie. The output is arranged decreasingly based on engagement score and being limited to the first 20 rows.

   Outputs : Title, Engagement Score

6. **List of reviews of a selected show**

   Justification : Users can read reviews based on the show they desire. Hence, a query that fetches the list of reviews is needed for the website. A function is created here so that a flexible query can retrieve a list of reviews for any given show.

   Tables involved : Up Vote, Down Vote, Review, Comment RLS, Movie, Series, Users

   Mechanics : A subquery is run first to filter the chosen show and fetch the corresponding show_id. A where condition in the main query filters the review ratings view to only include reviews from a given show_id. The output table is then joined to the users table through user_id to produce the final output table.

   Outputs : User ID, Name, Email Address, Review, Ratings, Number of Upvotes, Number of Downvotes and Number of comments

7. **List of actors of a selected show**

   Justification : This information is displayed when users are looking for show details.

   Tables involved : Movie, Series, Actors, Actor RLS

   Mechanics : A subquery is run first to filter the chosen show and fetch its corresponding show_id. The show_id is then used to find the actor-movie relationships in the Actor RLS table. Only actors with the show_id in Actors RLS is chosen from the list of all actors available in the actor table.

   Outputs : Actors ID, First Name, Last Name, Gender, Origin, Birth date

8. **Update cell and Insert data**

   Justification : A function is created so that the website system can update changes to be made in any table stored in the database. The changes include updating existing data in and inserting new data into the database