Database System Design for Online-Video Platforms

119010181 Xinhao LIN
119020054 Xinyang WANG
119020010 Muhan GU
119020071 Haotian ZHAI
119020019 Zhijun HUANG

SPRING ‖ 2022

CSC 3170
Database System

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Contents

1 Introduction and Motivation

This project is designed to support the operations of online-video platforms. More specifically, our target client is Disney+, which is an American subscription video streaming service owned and operated by the Media and Entertainment Distribution Division of the Walt Disney Company. It was first launched in November 2019, only a month before the start of the COVID-19 Pandemic.

The COVID-19 Pandemic has caused an explosion of the Internet use rate. Since people across the world tend to stay at home, they have hardly anything to do except surfing the Internet. According to the data of Statista[1], the subscribers of Disney+ grew from 26.5 million to 137.7 million throughout the pandemic. It seems that as many as four times the original users have crowded into this online video platform, bringing a huge amount of user data into the system.

Figure 1: Target Client: Disney Plus

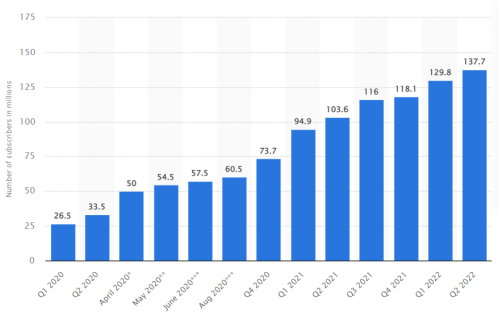

Figure 2: Growing Trend of Disney+ Users

Figure 3: Database Construction

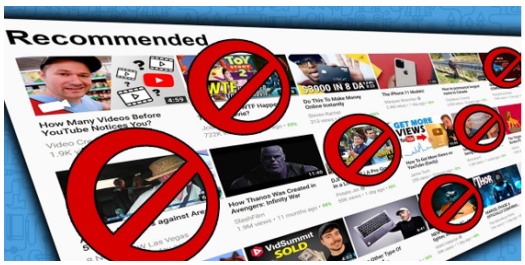

Figure 4: Recommendation System

To logically handle the exploding amount of data, our team would like to propose a database construction plan to organize all the platform users, online content, and related information. Additionally, our service will also include accurately identifying each user's preference by conducting data mining and building up a recommendation system, so that all the users can quickly find what they like most.

2 Design and Implementation

In this section, we focus on the design of the overall database system. To guarantee the feasibility and reduce the complexity of the database, we first create detailed business

rules, which can be found in Appendix (section 7.1). Then, we design an entity-relationship diagram for the database and convert it into a relational schema. Last but not least, we populate the schema with realistic data and provide some sample SQL queries for practical daily operations.

2.1 Basic Structure Design

2.1.1 Building up Entities and Relationships

To establish the database from the very beginning, we first build up the most important two entities: Users and Shows, which contain the basic information of the users and shows of the platform. The simplest activity that users do on the platform is watching shows, which naturally generates the Histories table. Besides watching, users may want to favorite some shows to pick up later, for which we design a Favorites table to record. What's more, users may want to give ratings and comments on the shows they watched, which can be collected in a Reviews table.

Considering the information of the Shows, we separate three tables from the main table: Actors, Genres, and Directors, in order to avoid redundancy. We then use three corresponding bridge tables to link them back to the main table. By doing so, transitive dependencies are removed at this very beginning of our design.

Lastly, since we are going to do recommendation later, we record the user preferences in Pref_Actor and Pref_Genre, which are also connected to the Actors and Genres tables.

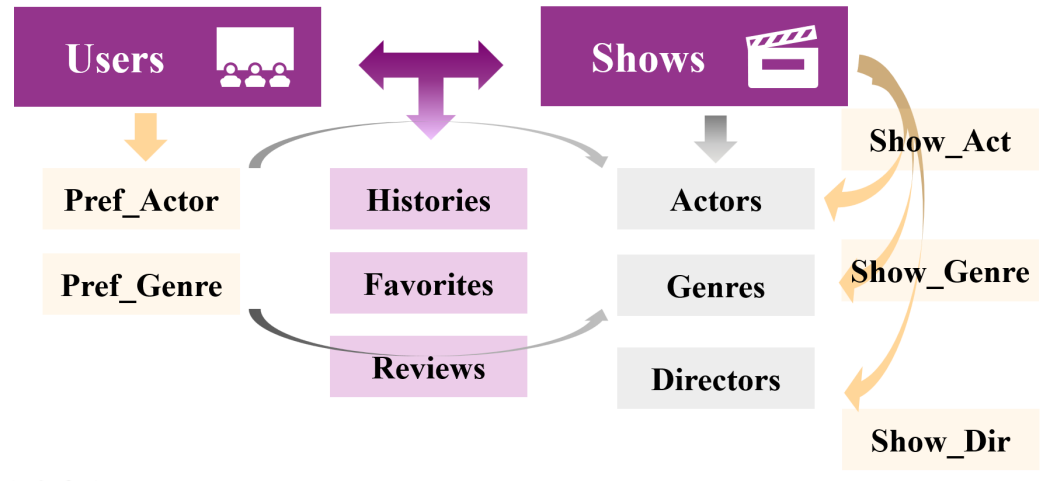

Figure 5: Overview of the Design Process

2.1.2 Entity-Relationship Diagram

The ER Diagram of our database system is shown below in Figure 6. It is in the style of crow's foot notation. In the graph, the primary keys have a yellow icon before them, the foreign keys have a blue icon before them, and those that are both primary keys and foreign keys have a red icon before them.

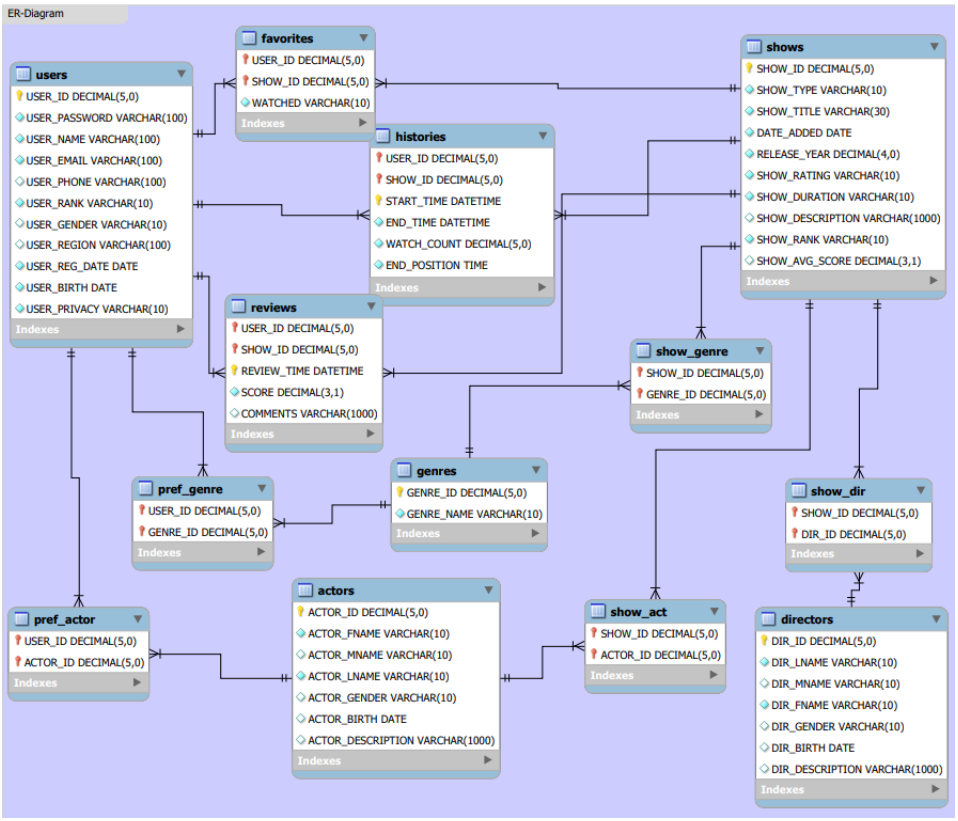

Figure 6: ER Diagram

2.1.3 Relational Schema

The relational schema of our database system is shown in Figure 7.

- Primary Keys: the **bold and underlined** attributes in the schema.
- Foreign Keys: the blue attributes in the schema.
- Functional Dependencies: all the functional dependencies on the primary keys are omitted in the schema (since they are obvious); the only functional dependency other than this is shown in the schema by the red arrows, that is, all the attributes in the USER table depends on user_email, which is a candidate key according to the business rules we set previously (therefore not causing transitive dependency).
- Multi-value Dependencies: there do not exist multi-value dependencies in our schema.

2.1.4 Normal Form

Our database design is in 4NF, since:

- there are no non-trivial functional dependencies of attributes on anything other than a superset of a candidate key (therefore satisfies BCNF)
- there are no multi-value dependencies (therefore satisfies 4NF)

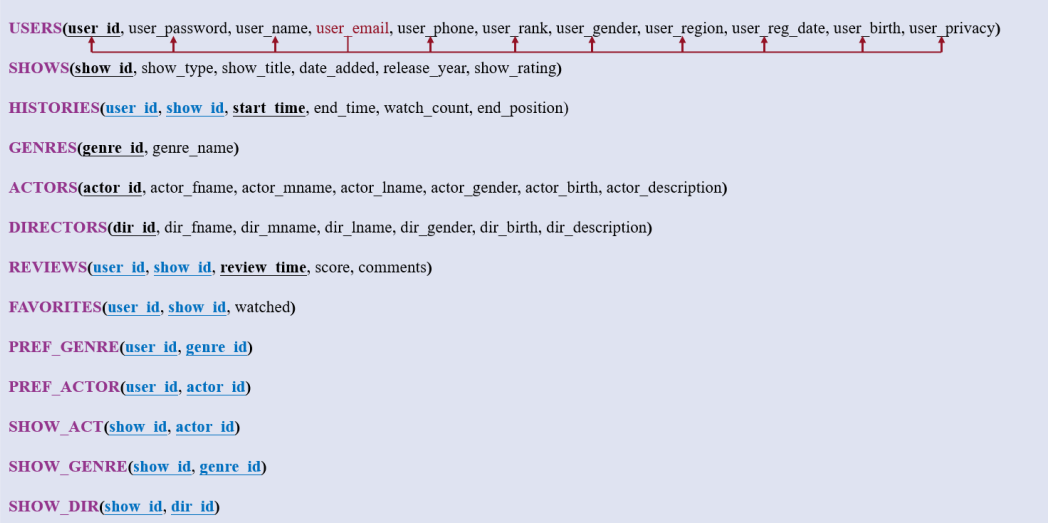

Figure 7: Relational Schema

This minimizes redundancy and improves the efficiency of the whole system.

2.1.5 Indexing

Based on our blueprint of the database system, we consider several data fields that are frequently used in data retrievals and manipulations, which are:

- USER_ID
- SHOW_ID
- ACTOR_ID
- DIR_ID
- GENRE_ID
- START_TIME
- REVIEW_TIME

After those data fields are indexed/hashed, it is expected that the number of disk accesses required can be minimized when a query is processed.

2.1.6 Important Attributes

The detailed descriptions and specifications of all attributes in the schema are appended in Appendix (section 7.2). Here we only introduce some important attributes that are vital to realizing the basic functions of the database:
Important attributes in the table SHOWS include:

- **show_rating:**

  This attribute is used to categorize shows by their content. There are 5 kinds of rating in total (requirements on children's age and parental guidance), which are used to regulate which group of people are allowed to watch the show.

- **show_rank:**

  Gives four levels of premiums. When registering, users are default to be in the silver level. Users can recharge to get to higher levels, which are sequentially gold, platinum, and diamond. The higher the rank is, the better accessibility users have to shows.

- **show_avg_score**

  Represents the average score of the show given by all users, which can be updated automatically by trigger (will be shown later) after each new score is given.

Important attributes in the table FAVORITES include:

- **watched:**

  Indicates whether the show has been watched by the user. If it is watched, the favorite show can resume from the last end position when clicked in.

Important attributes in the table USERS include:

- **user_birth:**

  Records birthday of the user. It is used to verify whether the user is qualified to watch certain shows.

- **user_privacy:**

  Controls whether the user's personal information is visible to others, including gender, rank, registration date, and birthday.

- **user_rank:**

  It has the same format as the show_rank attribute in the table SHOWS. Users with rank higher than or equal to that of the show can allowed to watch the show.

2.2 Database Implementation

Our database implementation contains the following two steps: the first step is to write a web crawler to get raw data and create database tuples based on it; the second step is to use functional queries to support daily operations and activities.

2.2.1 Schema Population

Based on The New York Times[2] and the official website of Disney+, we wrote a web crawler to get information about over 1400 movies and TV shows following the steps shown below:

- Borrow several Disney+ accounts
- Store the username and password in a dictionary, then use urllib to get the encoded dictionary
- Simulate login using the encoded dictionary and get cookies using the request library
- Use get() in request library to get HTML data. The parameters in get() consist of the URL, proxy and header
- Use the bs4 library to parse the HTML data
- Get the HTML tags of the content to be crawled by examining the web, and then use the tags to get the required data

We then implement different information padding methods for different tables. For table "USERS", we manually generate tuples to ensure the variety of data. For relationship tables such as the one between shows and actors, we tried to use JavaScript programs to implement auto filling of data.

2.2.2 Functional Queries

We have implemented four types of core functional queries as follows:

- INSERT: user registration and shows on shelves
- DELETE: user withdraw and shows off shelves
- UPDATE: change of users' rank when recharging
- SELECT: closely related to GUI; extract desired information such as users' favorite movies and shows, movies' actors and directors

We demonstrate part of the basic functional queries here in figure 8:

```
#register a user
INSERT INTO USERS
VALUES (79, 'Justin999', 'Highlight', 'justinGaetjche@gmail.com', '83752941', 'diamond', 'Male', 'US', '2020-01-11', '1995-01-01', 'TRUE');
#update a user's rank
UPDATE USERS
    SET USER_RANK = 'diamond'
    WHERE USER_ID = 11;
#delete a show
DELETE FROM SHOWS
WHERE SHOW_ID = 93;
```

Figure 8: Demo Queries

Besides these fundamental functional queries, we also implemented some advanced techniques.

We used the command "ON DELETE CASCADE" when writing tables, so that referential-integrity constraint is automatically guaranteed when we delete tuples. For example, we add "ON DELETE CASCADE" in table "HISTORIES". So when a user withdraws, his/her corresponding viewing histories will also be deleted.

```
CREATE TABLE HISTORIES
    (USER_ID          DECIMAL(5,0),
     SHOW_ID          DECIMAL(5,0),
     START_TIME       DATETIME,
     END_TIME         DATETIME     NOT NULL,
     WATCH_COUNT      DECIMAL(5,0)     NOT NULL,
     END_POSITION     TIME     NOT NULL,
     PRIMARY KEY (USER_ID, SHOW_ID, START_TIME),
     FOREIGN KEY (USER_ID) REFERENCES USERS(USER_ID) ON DELETE CASCADE,
     FOREIGN KEY (SHOW_ID) REFERENCES SHOWS(SHOW_ID) ON DELETE CASCADE);
```

Figure 9: On Delete Cascade in Table "HISTORIES"

Here is an example in figure 9:

We also implemented a trigger to automatically update a show's average score when a new user review is added. Triggers can help us save time and manual operations.

The implementation of trigger is shown in figure 10:

```
-- Trigger for automatic update of SHOWS.SHOW_AVG_SCORE --
DROP TRIGGER IF EXISTS trg_avg_score;
CREATE TRIGGER trg_avg_score
AFTER INSERT ON reviews

FOR EACH ROW
UPDATE SHOWS

SET SHOWS.SHOW_AVG_SCORE = (SELECT AVG(REVIEWS.SCORE)
FROM REVIEWS
WHERE REVIEWS.SHOW_ID = NEW.SHOW_ID)
WHERE SHOWS.SHOW_ID = NEW.SHOW_ID;
-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ --
```

Figure 10: Trigger

With the combination of basic functional queries and advanced techniques, our database is now complete and convenient in practical use.

3 Data Mining

In this section, we focus on extracting and discovering the patterns in the data using simple SQL queries and advanced machine learning techniques. First, we try to explore patterns and discover knowledge in our database. Then, we try to build a recommendation system model based on the classic "MovieLens" data set, since our data is not large enough for model construction. This recommendation model can then be further used on any realistic data in the future, including those in our Disney+ database.

3.1 Knowledge discovery

In this section, we want to discover some interesting patterns based on our own data.

3.1.1 Most popular movies

We want to find the most popular movies among the users in our database, so we use Python to scan through the movie names and get the following Word Cloud:

Figure 11: Most Popular Movies

The movies with larger names in the graph are the ones that are more popular. From the graph, we can see that there are two extremely popular movies in our data: "The Godfather" and "The Avengers".

3.1.2 Movie Clusters

We also want to find, if one user likes one Movie, what other movies he or she may also like. Here, we use "Movie Clusters" to denote such concept. Some results are illustrated below. If a user likes "The Avengers", he or she is very likely to like "Mission Impossible".

Figure 12: The Avengers

Figure 13: Mission: Impossible

3.1.3 Actor Clusters

Similarly, we also want to find, if one user likes an actor, what other actors he or she may also like. Here, we use "Actor Clusters" to denote such concept.

The figure below illustrates our result. We can see that, if one user likes Caption America, he or she may also like the Iron Man, Black Widow, Thor, Hulk, and the Spider Man. Most of them all appear in the Movie "The Avengers" and are closely related to Caption America. However, the Spider Man is not. One possible explanation is that, he is

closely connected with the Iron Man, since the user who likes Caption America also likes the Iron Man, he or she is very likely to like the Spider Man.

Figure 14: Actor Clusters

3.2 Movie Recommendation

3.2.1 Introduction to Recommendation

Recommendation systems aim to predict users' preferences and recommend product items that are likely to be interesting to them. Data required for recommendation systems stems from explicit user ratings after watching a movie, implicit search engine queries, and other knowledge about the users/items themselves. For example, if we want to recommend books for users who give explicit feedback, we have the following matrix:

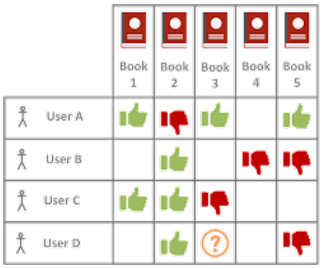

Figure 15: Illustration on Recommendation

3.2.2 Solving Recommendation Problem

Our problem is similar to the one illustrated above. In our case of Movie recommendation, we have ratings in matrix entries. Then, we can do recommendation by solving a matrix factorization problem.

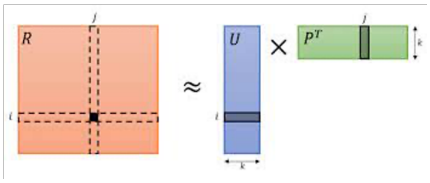

Figure 16: Solving Matrix Factorization

We aim to provide personalized recommendations beyond simple popular recommendations like the ones below:

Figure 17: Popular Movie Recommendation

Figure 18: Popular Animated Movie Recommendation

3.2.3 Model Implementation

Here, we try two models from the "Microsoft Recommenders"[3] to solve the aforementioned matrix factorization problem: Alternative Least Squares and Singular Value Decomposition.

For Alternating Least Squares (ALS), we want to find latent factors that represent intrinsic user and "item" (in our case, user and movie) attributes in a lower dimension. That is,

$$r_{u,i} = q_i^T p_u$$

where $r_{u,i}$ is the predicted ratings for user u and item i, and q_i^T and p_u are latent factors for item and user, respectively. Furthermore, to avoid over fitting issue, the learning process is regularized. For example, a basic form of matrix factorization algorithm is represented as below:

$$\min \sum (r_{u,i} - q_i^T p_u)^2 + \lambda (||q_i|| + ||p_u||)^2$$

Since this loss function is non-convex, though gradient descent method can be applied but this will incur expensive computations. As a result, Alternating Least Square (ALS) algorithm was used to overcome this issue. The basic idea of ALS is to learn one of q and p at a time for optimization while keeping the other as constant. This makes the objective at each iteration convex and solvable. The alternating between q and p stops when there is convergence to the optimal.

Then, for Singular Value Decomposition (SVD), it's quite similar to ALS, but with slight modifications of adding bias terms for user and item. Therefore, the model of SVD is as follows:

$$r_{u,i} = \mu + b_u + b_i + q_i^T p_u$$

where b_u and b_i are the user bias and item bias, and μ is the global average of all the ratings in the data set. The regularized optimization problem naturally becomes:

$$\min \sum (r_{u,i} - (\mu + b_u + b_i + q_i^T p_u))^2 + \lambda(\|b_u\|^2 + \|b_i\|^2 + \|q_i\|^2 + \|p_u\|^2)$$

Then, Stochastic gradient descent (SGD) is used for optimization.

After experiments, the results are summarized as below:

Metrics	**ALS**	**SVD**
Training Time(s)	4.278	10.344
Testing Time (s)	27.527	12.061
RMSE	0.9651	0.9488
MAE	0.7530	0.7470
MAP@3	0.339%	0.941%
MAP@10	0.653%	1.562%
Precision@3	4.813%	12.29%
Precision@10	5.172%	10.04%
Recall@3	0.519%	1.275%
Recall@10	1.884%	3.527%

Table 1: Recommendation Result

As is illustrated in the table, SVD generally outperforms ALS, but it requires a longer training and testing time, which is probably due to the added bias terms. To build up an accurate recommendation system, SVD shall be chosen as our final model for recommendation tasks. For more details about the evaluation metrics, please see the Appendix (section 7.3).

4 GUI Implementation

To make our database user-friendly, we designed a web-app demo. It didn't cover all the functionalities as we demonstrated above, but has implemented most of the selecting and searching functions. Here is our GitHub repository of this project's front-end and web server: Demo.

4.1 Technology Stack

We used HTML, CSS, JavaScript to implement the user interface, which is presented as a website. Moreover, we employed "amis", which is a low code front-end framework that uses JSON configuration to generate pages, reducing page development effort and greatly improving efficiency. For the back-end support, we used Express in Node.js to set up middlewares to respond to HTTP requests.

4.2 Functionality Implementation

- The home page includes a slideshow of the news and other information as Disney Plus. It also indicates our project's name and other basic information.
- Users can access six index pages via clicking on the tabs lying on the left bar of website. Each of them listed the elements in the corresponding tables.
- By clicking on each elements in the index pages, users can jump to the detail pages which shows more detailed information and related elements.
- In each detail page, users can jump to any other related detail pages by clicking on the elements occurred in the page.
- Though our schema doesn't include attributes storing images, we inserted some sample pictures in the website for the sake of beauty.

4.3 Screen Images

These are the sql queries required in implementing the following two pages.

```
select * from shows
order by SHOW_ID limit ${limit} offset ${offset};
select count(*) from shows;
select * from shows
where SHOW_ID =${movie_id};
select * from actors as a inner join show_act as b
on  a.ACTOR_ID=b.ACTOR_ID
where b.SHOW_ID = ${movie_id};
select * from directors as d inner join show_dir as b
on  d.DIR_ID=b.DIR_ID
where b.SHOW_ID = ${movie_id};
select u.USER_ID, u.USER_NAME,
c.SHOW_ID, c.SCORE, c.REVIEW_TIME, c.COMMENTS
from reviews as c inner join users as u
on c.USER_ID=u.USER_ID where c.SHOW_ID= ${movie_id};
select * from genres as g
inner join show_genre as s
on g.GENRE_ID = s.GENRE_ID
where s.SHOW_ID = ${movie_id};
```

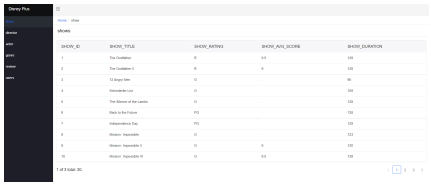

Figure 19: Index page listing all the shows

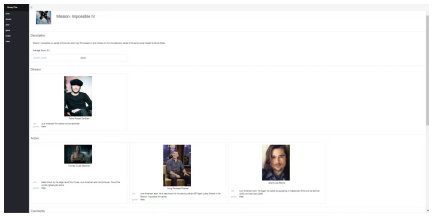

Figure 20: Detail page of a specific show

5 Conclusion and Self-evaluation

5.1 Conclusion

In this project, we have completed a thorough database design, function implementation, and data mining for Disney+. The entire process can be summarized as follows:

1. Analyze the background and requirements of Disney+; identify relevant entities, relationships, and the corresponding constraints; determine the business rules
2. Produce the E-R diagram, the relational schema, and the detailed descriptions for the database
3. Optimize the database operations by specifying the attributes that need to be indexed
4. Populate the schema with reasonable amount of realistic data; produce sample SQL queries to realize the basic functions of the database
5. Conduct data mining and propose a recommendation system to help with the customer service and preference analysis of Disney+
6. Build a Graphic User Interface to visualize the database system

5.2 Self-evaluation

In this project, the division of labor is reasonably equal among all the team members (20% for each member). The details are as follows:

- The design of all aspects of the project is collaboratively done by all members.
- The visualization & presentation of the conceptual part is distributed to Zhijun HUANG (119020019) and Xinyang WANG (119020054).
- The realization & presentation of the functional part is distributed to Haotian ZHAI (119020071), Muhan GU (119020010), and Xinhao LIN (119010181).

6 References

[1] Statista. (2022, May). *Number of Disney Plus subscribers worldwide from 1st quarter 2020 to 2nd quarter 2022(in millions).* Retrieved May 10, 2022, from https://www.statista.com/statistics/1095372/disney-plus-number-of-subscribers-us/

[2] The New York Times. (2022, April). *The 50 Best TV Shows and Movies to Watch on Disney+ Right Now.* Retrieved May 10, 2022, from https://www.nytimes.com/article/best-tv-shows-movies-disney-plus.html

[3] Microsoft Recommenders. (2022, April). *Recommenders 1.1.0.* Retrieved May 10, 2022, from https://github.com/microsoft/recommenders

7 Appendix

7.1 Business Rules

Rules related to USERS:

- USER_EMAIL must be provided and verified when users sign in, so that the platform can track the users' identities, while USER_PHONE is optional and users can add their phone numbers to further secure their accounts.
- USER_PRIVACY controls the visibility of USER_RANK, USER_GENDER, USER_REG_DATE, and USER_BIRTH. If it is set to be "True", then all of the above information is visible to all other users; if it is set to be "False", then all of the above information is hidden from all other users.
- USER_NAME is the only information that is always visible to all other users regardless of the value of USER_PRIVACY. Besides, identical usernames among different users are allowed.

Rules related to SHOWS:

- SHOW_RATING must be provided when the show goes public. There are 5 levels of rating for both TV shows and movies:

  1) G: General Audiences. All Ages Admitted.
  2) PG: Parental Guidance Suggested. Some Material May Not Be Suitable for Children.
  3) PG-13: Parents Strongly Cautioned. Some Material May Be Inappropriate for Children Under 13.
  4) R: Restricted, Children Under 17 Require Accompanying Parent or Adult Guardian.
  5) NC-17: No One 17 and Under Admitted.

  The rating provides a mechanism to protect children's mental health and guideline on what parents are allowing their children to watch.
- SHOW_RANK contains "silver", "gold", "platinum", and "diamond" (ascending order). Users with premium levels higher than or equal to the required rank have access to the show.

Rules related to HISTORIES:

- WATCH_COUNT is added by one when the user opens a web page of a show, watch for a while, and then close it.

Rules related to PREFERENCES, GENRES, ACTORS:

- Each user may prefer one genre, multiple genres, or no genres. A genre can contain one actor, multiple actors, or no actor. An actor can play one genre, multiple genres, or no genre.

Rules related to REVIEWS:

- When giving reviews, a user may only give the SCORE value and leave the COMMENTS part empty, or provide both the SCORE and the COMMENTS.

Rules related to SHOW, SHOW_GENRES, GENRES:

- Each show may have one genre, multiple genres, or no genres. A genre can correspond to one show, multiple shows, or no shows (When a show is taken off the shelves, and causes a genre having no shows to correspond to, the genre will not be removed from GENRES).

Rules related to SHOW, SHOW_DIR, DIRECTORS:

- Each show may have multiple directors, and must have at least one director. A director can direct one show, multiple shows, or no shows. (When a show is taken off the shelves, and causes a director having no shows to correspond to, the director will not be removed from DIRECTORS).

Rules related to SHOWS, SHOW_ACT, ACTORS:

- Each show may have multiple actors, and must have at least one actor. An actor can act in one show, multiple shows, or no shows. (When a show is taken off the shelves, and causes an actor having no shows to correspond to, the actor will not be removed from ACTORS).

Rules related to FAVORITES:

- Each user may favorite one show, multiple shows, or no shows. A show may be liked by any number of users.
- WATCHED indicates whether the show has been watched by the user who favorites it. If WATCHED is True, we can search the table HISTORIES to find the record of this user watching this show.

7.2 Attributes Description

USERS: This is a table that contains the basic information of any user of Disney+

- USER_ID: The user's identifier that determines other attributes. It is the primary key.
- USER_PASSWORD: The user's password to login the account. It must not be empty.
- USER_NAME: The username determined by the user when they create their account. It must not be empty.
- USER_EMAIL: The email address of the user. It must not be empty as it is used to secure the user's account (and the suffix can be omitted), and it must be unique.

- USER_PHONE: The phone number of the user. Might be empty. The character '-' Is allowed for a phone number. It must be unique.
- USER_RANK: The premium level of the user. Must not be empty. It should be one of silver, gold, platinum, diamond.
- USER_GENDER: The gender of the user. Might be empty. If the user specifies the gender, it should be one of Male, Female, Others.
- USER_REGION: The region of the user. Might be empty.
- USER_REG_DATE: The date when the user created the account for the first time. Must not be empty.
- USER_BIRTH: The birth date of the user. It must not be empty as it will be used to verify whether the user is old enough to watch some restricted shows.
- USER_PRIVACY: Whether the user's personal information is visible to others, which is specified by the user. It should be one of True, False, where 'True' means visible and 'False' means hidden from other users. It must not be empty.

SHOWS: This table contains information of all shows produced by Disney+

- SHOW_ID: The show's identifier that determines other attributes. It is the primary key.
- SHOW_TYPE: The type of the show, which should be one of TV Show, Movie. Must not be empty.
- SHOW_TITLE: The title of the show. Must not be empty.
- DATE_ADDED: The date of the show when it is on screen. Must not be empty.
- RELEASE_YEAR: The released year of the show. Must not be empty.
- SHOW_RATING: The rating of the show, which is used to restrict certain users' access to some movies (e.g. For movies with rating R, children under 17 require accompany parent or adult guardian). Must not be empty.
- SHOW_DURATION: The duration of the show. For TV shows, they are measured by the number of seasons, and movies are measured by the number of minutes. Must not be empty.
- SHOW_DESCRIPTION: The subject or plot of a show. Might be empty.
- SHOW_RANK: Describe the premium level which is required to watch the show. Have the same format as USERS.USER_RANK. Must not be empty.
- SHOW_AVG_SCORE: Describe the average score given by users. Have the same format as REVIEWS.SCORE. Might be empty.

HISTORIES: This table contains information of uses' watching histories

- USER_ID: The user id of a specific watching history. It is part of the primary key. Have the same format as USERS.USER_ID.
- SHOW_ID: The show id of a specific watching history. It is part of the primary key. Have the same format as SHOWS.SHOW_ID.
- START_TIME: The time slot when a user starts watching a show, which should be precise to seconds. It is part of the primary key.
- END_TIME: The time slot when users stop watching a show, which should be precise to seconds. Must not be empty.
- WATCH_COUNT: The number of times that a user watches the same show. Must not be empty.
- END_POSITION: The position of the show when users stop watching. Must not be empty.

GENRES: Describe the genres of shows in Disney+

- GENRE_ID: The show's identifier that determines other attributes. It is the primary key.
- GENRE_NAME: The name of the genre. Must not be empty. Must be unique.

ACTORS: This is a table that contains the basic information of all the actors in the Disney+ shows

- ACTOR_ID: The actor's identifier that determines other attributes. It is the primary key.
- ACTOR_FNAME: The actor's first name. Must not be empty.
- ACTOR_MNAME: The actor's middle name. Might be empty.
- ACTOR_LNAME: The actor's last name. Must not be empty.
- ACTOR_GENDER: The gender of the actor. Might be empty. If the gender is specified, it should be one of Male, Female, Others.
- ACTOR_BIRTH: The birth date of the actor. Might be empty.
- ACTOR_DESCRIPTION: A descriptive passage of the actor. Might be empty.

DIRECTORS: This is a table that contains the basic information of all the directors in the Disney+ shows

- DIR_ID: The director's identifier that determines other attributes. It is the primary key.
- DIR_FNAME: The director's first name. Must not be empty.
- DIR_MNAME: The director's middle name. Might be empty.

- DIR_LNAME: The director's last name. Must not be empty.

DIR_GENDER: The gender of the director. Might be empty. If the gender is specified, it should be one of Male, Female, Others

- DIR_BIRTH: The birth date of the actor. Might be empty.
- DIR_DESCRIPTION: A descriptive passage of the director. Might be empty.

REVIEWS: This is a table containing the ratings and comments on the Disney+ shows given by the users

- USER_ID: Identifies the user who gives the rating and comments. It is part of the primary key. Have the same format as USERS.USER_ID.
- SHOW_ID: Identifies the show being reviewed. It is part of the primary key. Have the same format as SHOWS.SHOW_ID.
- REVIEW_TIME: A timestamp identifying the time when the review is given. It is part of the primary key.
- SCORE: The rating score given by the user. It should be a value between [1, 10], and it is either an integer or a floating number with .5 behind the integer part. Must not be empty.
- COMMENTS: A passage given by the users to comment on the show. Might be empty.
- FAVORITES: This is a table recording the shows favored by each user
- USER_ID: Identifies the user by whom the show is favored. It is part of the primary key. Have the same format as USERS.USER_ID.
- SHOW_ID: Identifies the show being favored. It is part of the primary key. Have the same format as SHOWS.SHOW_ID.
- WATCHED: A value in "True", "False" that specifies whether the show being favored has been watched by the user. Must not be empty.

PREF_GENRE: This table contains users' preferences on genres

- USER_ID: Have the same format as USERS.USER_ID.
- GENRE_ID: Have the same format as GENRES.GENRE_ID.

PREF_ACTOR: This table contains users' preferences on actors

- USER_ID: Have the same format as USERS.USER_ID.
- ACTOR_ID: Have the same format as ACTORS.ACTOR_ID.

SHOW_ACT: This is a bridge table connecting SHOWS and ACTORS

- ACTOR_ID: The actor's identifier. It is part of the primary key. Have the same format as ACTORS.ACTOR_ID.

- SHOW_ID: The show's identifier. It is part of the primary key. Have the same format as SHOWS.SHOW_ID.

SHOW_DIR: This is a bridge table connecting SHOWS and DIRECTORS

- DIR_ID: The director's identifier. It is part of the primary key. Have the same format as DIRECTORS.DIRECTOR_ID.
- SHOW_ID: The show's identifier. It is part of the primary key. Have the same format as SHOWS.SHOW_ID.

SHOW_GENRE: This is a bridge table connecting SHOWS and GENRES

- GENRE_ID: Identifies the genre of the show. It is part of the primary key. Have the same format as GENRES.GENRE_ID.
- SHOW_ID: Identifies the show. It is part of the primary key. Have the same format as SHOWS.SHOW_ID.

7.3 Recommending System Evaluation Metrics

Top K Measurement

- Since the project aims to do recommendations for the users, the focus of the evaluation is on the top user ratings that our models predict, i.e., the following evaluation metrics we introduced will mainly be conducted on the top-k testing results, with k selected differently on different models. That is, for example, the precision for the top-k testing results, metric is called "precision@k".

RMSE (Root Mean Square Error)

- Definition: A metric that tells us the square root of the average squared difference between the predicted values and the actual values in a data set.
- Evaluation criterion: The lower the RMSE, the better a model fits a data set.
- Formula:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}}$$

MAE (Mean Absolute Error)

- Definition: A metric that tells us the mean absolute difference between the predicted values and the actual values in a data set.
- Evaluation criterion: The lower the MAE, the better a model fits a data set.
- Formula:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Precision

- Definition: the number of correct results divided by the number of all returned results (that is, the correct portion of positive predictions).
- Evaluation criterion: The higher the precision, the better a model fits a data set.
- Formula:

$$Precision = \frac{N_{TruePositive}}{N_{TruePositive}+N_{FalseNegative}}$$

Recall

- Definition: the number of correct results divided by the number of results that should have been returned (that is, the correct prediction rate of positive samples).
- Evaluation criterion: The higher the recall, the better a model fits a data set.
- Formula:

$$Recall = \frac{N_{TruePositive}}{N_{Positive}}$$