# PennliPennli

Application for anime lover

Liwen Xie, Ziyao Chen, Cong Cao, Xinyu Liu

# Contents



# 1 Introduction

## 1.1 Project goals

In this application, we will build a multifunctional web application that allows users to quickly retrieve information of synopsis, genre, scores, and reviews of animes. This application will also provide services for anime lovers, like ranking anime with specific genres, creating their own favorite watching list and searching for similar anime they have watched base on genre.

### 1.2 Application functionality

Our application mainly has three functions. Firstly, you can search anime by key word, and filter the result by genre, score, ranking, popularity,episodes. Secondly, we can display fascinating animes info to our user, show similar anime based on current anime genre, and display review about the anime. Last but not least, user can click on other user's name and find which anime other user like and recommend animes based on current user's favorite list' genre.

### 1.3 Motivation

Our web is designed to show user animes and convenient functionality. Once user click anime' image, it will bring user to the information part and all the reviews. By using our web, our user can explore similar animes, find other users who reviewed the anime and their favorite list. We hope user get attention by our recommendation and find animes that they did not watch before.

## 2 Architecture

### 2.1 Technologies used

**Front-end**: React
**Back-end**: Python for data pre-processing, Node.js and React
**Database**: MySQL hosted on AWS

### 2.2 System architecture

Our application follows the similar architecture as Homework 2, with React on the front-end, Node and MySQL RDS database for back-end. We also installed several packages to help with routing and CSS styling, such as react-dom, react-router, react-router-dom, react-scripts, react-slick, shards-react, slick-carousel, tachyons, web-vitals.

## 3 Data

### 3.1 Data Source

We will use the "Anime Dataset with Reviews -My AnimeList" datasets from kaggle. This dataset contains information about Anime (16k), Reviews (130k) and Profiles (47k) crawled from https://myanimelist.net/ at 05/01/20.

### 3.2 Database Tables

The dataset contains 3 files, which can be found from
https://www.kaggle.com/marlesson/myanimelist-dataset-animes-profiles-reviews:

- animes.csv contains a list of anime, with title, title synonyms, genre, duration, rank, popularity, score, airing date, episodes and many other important data about individual anime providing sufficient information about trends in time about important aspects of anime. Rank is in float format in csv, but it contains only integer values. This is due to NaN values and their representation in pandas.

- profiles.csv contains information about users who watch anime, namely username, birth date, gender, and favorite anime list.

- reviews.csv contains information about reviews users x anime, with text review and scores.

### 3.3 Relevant summary statistics

| Brief Summary Statistics | | | |
|---|---|---|---|
| Tables | Name | Mean | Standard Deviation |
| 1*Reviews | score | 7.97 | 2.08 |
| 3*Anime | score | 6.90 | 1.06 |
| | popularity | 5504.53 | 3717.45 |
| | members | 70280.46 | 173581.25 |

### 3.4 Data cleaning and processing

We used Python Pandas to read our raw CSV files.

- Drop rows which keys are empty
- Extract **genre** from **profiles.csv**, which is originally formed as Lists, use set operation to get the final set of **genre**, which will be used in DDL.
- Extract **favorites animes** from **profiles.csv**, which is originally formed as Lists, we would extract each element from each list and map them with corresponding Name, and store as a new CSV file named as **Favorites.csv** in the format of (Name, favorite anime)
- Remove unused column such as **Scores** in **reviews.csv**
- Remove the heading useless line of each review in **reviews.csv**

### 3.5 Dataset relationship

In the Anime dataset, there is a primary key*Anime_uid* that linked to the other datasets.
In the Favorites dataset, there are two foreign keys: Name that linked to the Name in the User dataset, and anime_id that linked to the Anime_Uid in the Anime dataset.
In the Genre dataset, there is a primary key (Anime_uid, Genre) and a foreign key Anime_uid that linked to the Anime_uid in the anime datasets.
In the User dataset, there is a primary key *Name* that linked to the other datasets.
In the Reviews dataset, there is a primary key Uid and two foreign keys: name that linked to the name in the User dataset, and Anime_Uid that linked to the Anime_Uid in the Anime dataset.

## 4 Database

### 4.1 Data ingestion procedure and entity resolution efforts

For the original data, User's **Name** (it is the user id name, which is unique), Anime's **Anime_id**, Reviews' **Uid** could all determine the tuple of it corresponding table. Hence we use these as primary keys for tables.
After data cleaning and pre-processing, we found there are several many-to-many relationships in our database, so we generated two new tables: Favorites and Genre, each contains Anime id-Name pair and Anime id-Genre pair. These two tables accelerate our search and recommendation.

## 4.2 Relational schema as an ER diagram

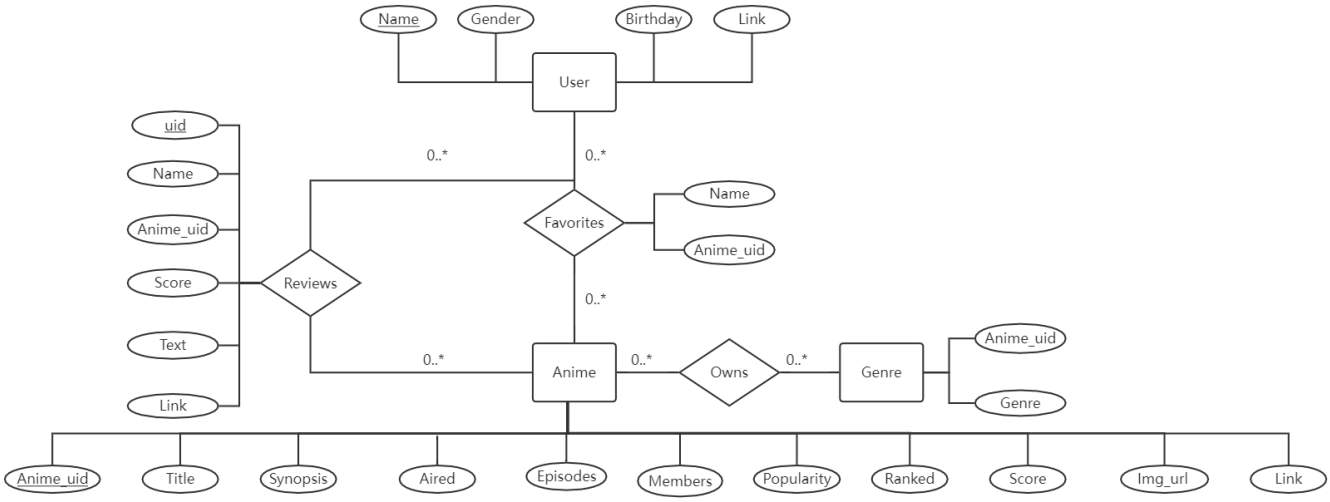


Figure 1: ER diagram

## 4.3 Statistics - Number of instances in each table

We build five tables, Anime, Favorite, Genre, Reviews and Users.
The Anime table contains 3157 instances; the Favorite table contains 32809 instances; the Genre table contains 8998 instances; the Reviews table contains 21006 instances; the User table contains 10379 instances.

## 4.4 Details on the normal form used

### Relational Schema

**User** (Name, Gender, Birthday, Link)
**Anime** (Anime_uid, title, Synopsis, Aired, Episodes,
Members, Popularity, Ranked, Score, Img_url, Link)
**Favorites** (Name, Anime_id)
Name **FOREIGN KEY REFERENCES** User(Name)
Anime_uid **FOREIGN KEY REFERENCES** Anime(Anime_uid)
**Genre** (Anime_uid, Genre)
Anime_uid **FOREIGN KEY REFERENCES** Anime(Anime_uid)
**Reviews** (Uid, Name, Anime_uid, Text, Score, Link)
Uid Primary Key
Name **FOREIGN KEY REFERENCES** User(Name)
Anime_uid **FOREIGN KEY REFERENCES** Anime(Anime_uid)

### Functional Dependencies

Listing the functional dependencies for each table as following:

**User**: Name -> Gender, Birthday, Link
**Anime**: Anime_uid -> title, Synopsis, Aired, Episodes, Members, Popularity,
Ranked, Score, Img_url, Link
**Favorites**: Name, Anime_id -> Name, Anime_id
**Genre**: Anime_uid, Genre -> Anime_uid, Genre
**Reviews**: Uid -> Name, Anime_uid, Text, Score, Link

According to the definition of BCNF,
For every relation scheme R and for every $X->A$ that holds over $R$, either $A \in X$(it is trivial), or $X$ is a super key for $R$.
We can conclude that all the tables are in BCNF (then in 3NF automatically) because the primary key of each table uniquely defines the remaining attributes of the tables, here primary key is the candidate key, which is minimal super key.

# 5 Web App description

- Home page: Default show top 100 popular anime in the form of anime card. Support keyword search functionality, filter based on user customized range of Episodes, Popularity, Score, Rank. Click on the picture of anime card will link to anime page with specified anime_uid.
- Anime page: Link from any anime card in Home, Anime, Recommendation Page, show the detail information such as Title, score, synopsis, user reviews of the specific anime, and corresponding recommended top 8 similar anime based on the genre of current anime. User reviews consist user name, the score given, and the text part of user's review, clients can click on each user's name to link to their favorite anime profile and corresponding recommendation.
- Recommendation page: From link in Anime page, show specific user's favorite anime list, and offer Recommendation based on current user's favorite anime list.
- About Us pages consist team member's information.

# 6 API Specification

- Route 1: /home
  Description: Home Page that displays top 100 anime based on popularity by default.
  Route Parameter(s): None
  Query Parameter(s): None
  Query: **Query 1 (from Part 7)**
  Route Handler: home_display(req, res)
  Return Type: JSON
  Return Parameters:
  { results (JSON array of { title (String), Anime_uid(Integer), Score (float), Img_url (String), Link (String) }) }
  Expected (Output) Behavior :
  1. Display Top 100 popular anime
     Return { Title: _, Anime_uid: _, Score: _, Img_url: _, Link: _}
  2. None
     Return { error: _}
- Route 2: /home/search
  Description: Home Page's search feature part, which displays anime that contains search keywords.
  Route Parameter(s): None
  Query Parameter(s):keyword ? '%' + keyword + '%' : '%'
  Query: **Query 2 (from Part 7)**
  Route Handler: home_search (req, res)

Return Type: JSON
Return Parameters:
{ results ( JSON array of { Title (String), Score (float), Anime_uid(Integer), Img_url (String), Link (String) } ) }
Expected (Output) Behavior :

1. Case 1: Display anime that contains search keywords.
   Return { Title: _, Episodes: _, Anime_uid:_, Img_url: _, Link: _ }
2. Case 2: None
   Return { error: _}

- Route 3: /search/animes
  Description: Anime filtering page to filter in and out anime based on specified genre, episodes range, score, ranked range
  Route Parameter(s): None
  Query Parameter(s): req.query.episodes_low, req.query.episodes_high, req.query.popularity_low, req.query.popularity_high, req.query.score, req.query.rankTop, req.query.rankBottom, req.query.gerne
  Query: **Query 3 (from Part 7)**
  Route Handler: search_animes (req, res)
  Return Type: JSON
  Return Parameters:
  {results (JSON array of { Title(String), Episodes (int), Ranked (int), Score (float), Img_url (String), Link (String) }) }
  Expected (Output) Behavior :

  1. Return an array with all the animes that match the constraints. If no anime satisfies the constraints, return an empty array as 'results' without causing an error.
  2. Alphabetically sort the results by (Popularity) attribute

- Route 4: /anime
  Description: Anime Page that shows detailed information about the anime, which is specified by anime_uid, such as synopsis, airing date, genre, episodes, reviews, user Name.
  Route Parameter(s): None
  Query Parameter(s): Anime_uid (int)
  Query: **Query 4 (from Part 7)**
  Route Handler: anime (req, res)
  Return Type: JSON
  Return Parameters:
  {results (JSON array of { Anime_uid(int), Title(String), Synopsis (String), Aired(String), Episodes (int), Members (int), Popularity (int), Ranked (int), Score (float), Img_url (String), Link (String), } ) }
  Expected (Output) Behavior :

  1. Display the detailed information of an anime, with specified anime_uid.

- Route 4.2: /anime/review
  Description: Reviews along with anime page, that shows detailed information about the anime reivews, which is specified by anime_uid.
  Route Parameter(s): None
  Query Parameter(s): Anime_uid (int)

Query: **Query 5 (from Part 7)**
Route Handler: anime_reviews (req, res)
Return Type: JSON
Return Parameters:
{results (JSON array of { Anime_uid(int), Name(String), Text(String), R.Link(String), User.Link(String) } ) }
Expected (Output) Behavior :

1. Display the detailed information of anime reviews, with specified anime_uid.

- Route 4.3: /anime/recommendation
Description: Shows similar anime recommendation for a specific anime, which are specified by anime_uid
Route Parameter(s): None
Query Parameter(s): Anime_uid (int)
Query: **Query 6 (from Part 7)**
Route Handler: anime_recommendation(req, res)
Return Type: JSON
Return Parameters:
{results (JSON array of { Anime_uid(int), Title(String), Score (float), Img_url (String) } ) }
Expected (Output) Behavior :

1. Display the recommended animes,score and pictures, with specified anime_uid.

- Route 5: /recommendation
Description: Recommendation page, it shows our anime recommendations based on the user's current favorite anime information.
Route Parameter(s): None
Query Parameter(s): userName
Query: **Query 7 (from Part 7)**
Route Handler: recommendation (req, res)
Return Type: JSON
Return Parameters:
{results (JSON array of { Title(String), Episodes (int), Popularity (int), Ranked (int), Score (float), Img_url (String), Link (String) } ) }
Expected (Output) Behavior :

1. Return an array with top 20 animes that are based on the user's current favorite anime information.

- Route 6: /favorite
Description: Favorite page, it shows the user favorite list with the maximum, minimum and average scores of each anime.
Route Parameter(s): None
Query Parameter(s): userName
Query: **Query 8 (from Part 7)**
Route Handler: favorite (req, res)
Return Type: JSON
Return Parameters:
{results (JSON array of { anime_uid (int), MinScore (float), MaxScore(float), AvgScore(float)} ) }
Expected (Output) Behavior:

1. Output the anime_uid, min, max, avg review scores of user' favorite anime

# 7 Queries and descriptions:

- Query 1: Order the anime by their popularity scores: return anime info such as Anime_uid, title, episodes, ranks, score, img_url, link. It is used to display anime info in the home page.

```
Select Title, Anime_uid, Score, Img_url, Link
From Anime
Order BY Anime.Popularity
Limit 100;
```

- Query 2: Search anime by the keywords and order the anime by their popularity scores: return animes info such as Anime_uid, title, score, img_url, link. It is used to implement search function in the home page.

```
Select Title, Anime_uid, Score, Img_url, Link
From Anime
Where Title like ‘ keyword ’
Order BY Anime.Popularity;
```

- Query 3: Search anime by the length of episodes, popularity, score, and rank ranges and order the anime by popularity: return animes info such as title, episodes, ranked, score, Img_url, link. It is used to implement search function in the home page.

```
SELECT DISTINCT Anime.Anime_uid, Title,Score, Img_url, Link
FROM Anime Join Genre on A.Anime_uid = Genre.Anime_uid
WHERE Episodes >= $episodes_low
AND Episodes >= $episodes_high
AND Popularity >= $popularity_low
AND Popularity <= $popularity_high
AND Score >= $score_low
AND Score <= $score_high
AND Ranked >= $rankHigh
AND Ranked <= $rankLow
AND Genre LIKE ‘$genre’
ORDER BY Popularity;
```

- Query 4: Search the specific anime by the anime id: return anime info such as anime_uid, title, episodes, ranked, score, Img_url, link.

```
SELECT Anime_uid, title, Synopsis, Aired, Episodes,
Members, Popularity, Ranked, Score, Img_url, Link
FROM Anime WHERE Anime_uid = $anime_uid;
```

- Query 5: Search the review of the specific anime by the anime id: return anime info and reviews such as anime_uid, Name, Score, Text, R.Link (scores), User.Link

```
SELECT R.Anime_uid, User.Name, R.Score, R.Text, R.Link, User.Link
FROM Reviews R JOIN User on R.Name = User.Name
WHERE R.Anime_uid = $anime_uid;
```

- Query 6: Search other top 8 recommended anime that are similar to a specific anime by the anime id: return Anime_uid, title, score, Img_url. It is used to display 8 similar animes in the anime page.

```
WITH RecomList AS
(
 SELECT DISTINCT Anime_uid
 FROM Genre
 WHERE
 Genre in
 (SELECT Genre FROM Genre
 WHERE Anime_uid = $anime_uid ))
 SELECT A.Anime_uid, A.title, A.Score, A.Img_url FROM
 Anime A JOIN RecomList R ON A.Anime_uid = R.Anime_uid
 WHERE A.Anime_uid <> $anime_uid
 ORDER BY A.Score DESC
 LIMIT 8;
```

- Query 7: Search for the top 20 anime recommendations (order by popularity) for a user according to the user's favorite anime list: return the anime info such as title, score, Img_url, Link and Anime_uid. It is used to display the recommendation for user list in the user page.

```
With Favorite_list
AS
(Select F.Anime_uid AS Anime_uid, G.genre AS genre
From Favorites F JOIN Genre G ON F.Anime_uid = G.Anime_uid
Where Name LIKE '$userName' )
Select  DISTINCT A.Title, A.Episodes, A.Ranked, A.Score, A.Img_url,
A.Link
From Anime A JOIN Genre G ON A.Anime_uid = G.Anime_uid
Where G.genre IN (Select DISTINCT genre From Favorite_list ) AND
A.Anime_uid NOT IN (Select DISTINCT Anime_uid From Favorite_list)
Order BY A.Popularity
LIMIT 20;
```

- Query 8: Search for a user's favorite anime list by username, calculate the maximum, minimum and average scores of the specific user: return title, MinScore, AvgScore, MaxScore, Img_url, Link and anime id. It is used to display Favorite Anime list in the user page.

```
With Favorite_list
AS
(Select F.Anime_uid AS Anime_uid, G.genre AS genre
From Favorites F JOIN Genre G ON F.Anime_uid = G.Anime_uid
Where Name LIKE '$userName')
Select R.anime_uid,
ROUND(MIN(score),2) AS MinScore,
ROUND(MAX(score),2) AS MaxScore,
ROUND(AVG(score),2) AS AvgScore
```

```
From Reviews R join Favorite_list Fl on R.Anime_uid = Fl.Anime_uid
Group BY anime_uid
Order BY Anime_uid ASC;
```

# 8 Performance evaluation

- **Adding Index**
  We added indexes on some tables in DataGrip. We improved our query 1 performance by adding indexes popularity_index on Anime.Popularity and index uid_index, which improving search time from 6.6s to 0.04s. To improving query 2, we added index title_index on Anime.Title, which improving time from 6.02s to 4.5s. The performance did not change much since our query search keyword by prefix and suffix and our query 2 query did not take full advantage from Index, but with index uid_index and popularity, its performance time ends up with 0.128s. With query 3 on adding index popularity, we improving time from 2.5 s to 0.89s. In Genre table, we add index on its genre attribute, which results time from 25.6s to 6.5s. Adding score_index on Anime table, the time changed from 0.59s to 0.35s for query 8. As a result, Anime's index(title_index, uid_index, score_index, popularity_index), Genre's index(genre_index).

- **Other optimizations: Splitting API**
  In the route 4, our original query gives detailed information about the anime, which is specified by anime_uid, such as synopsis, airing date, genre, episodes, reviews, user Name. It integrates three tables: **Anime, Reviews, Users**. We tested API at Postman and found out the fetch time is around 1040 ms. We found out that the Anime's reviews size are too big and resulted slow time page loading. We decided to split the API into two small APIs(along with a third newly created API for similar Animes), so that our page could show user some basic information about Anime along with the similar genre anime recommendation at first place and corresponding users' reviews. By doing that, we achieved faster page response by 173ms and better user experience.

- **Adding Cache**
  Adding cache solution can also improve our home page performance since every time user refresh page, it will fetch route 1's result. We want to implement it though MySQL, but MySQL 8.0 do not support query cache. We did not implement cache through server side due to time limit.

# 9 Technical challenges

The first challenge we encountered is cleaning the raw data sets. While dropping rows with empty keys, extract List form information were relatively easy with Python Pandas, processing the review text is hard and time consuming since there are non-uniform unless information in the beginning of each reviews. Non-uniformity of the reviews makes it hard to standardize the data easily, we could only try to process more than 190,000 reviews, each letter by letter to remove the heading part of reviews.

The second challenge is the fetcher part. It took us a while to figure out how to get the dynamic query parameter. Since React Hook cannot be used inside the class based Component, we made several approaches with **React Hook**, like wrap **useLocation**, **useparams**, in a function and pass the properties along. Unfortunately, none of these approaches works properly, our final solution is to use **window.location.search.substring**

Another thing is, for our anime and recommendation page, we integrated several functionality on the same page. And for the purpose of optimize the running time of queries, we implemented query optimization with lots of approaches as feasible and we found out that it is really hard to improve the running time of some query with one complex query, so we decided to split some APIs involved lots of **"Integrations"** into multiple faster simpler APIs and we spent time on figuring out how to use several APIs on the same page.