CSCI620 - Introduction to Big Data

Project Phase II
Group 1

Topic: MyAnimeList Data Analysis

Sources:
- Kaggle: https://www.kaggle.com/datasets/azathoth42/myanimelist

- MyAnimeList: https://myanimelist.net/

Authors – Athina Stewart (as1986)

Archit Joshi (aj6082)

Chengzi Cao (cc3773)

Parijat Kawale (pk7145)

Contents

1. Document-Oriented Model

- The document-oriented design model proposed is based on MongoDB.
- The general schema for MongoDB is as follows:
  - Anime collection:
    - _id : Unique anime id associated with each anime
    - Title: the actual title of the anime
    - Title_english: The english equivalent of the title
    - Title_japanese: The japanese equivalent of the title
    - Title_synonyms: Synonyms of the current title
    - Title_episodes: Number of episodes in each anime
    - Date_aired: date on which the anime aired
    - Is_airing: is the anime currently aired
    - Rank: current rank of the anime amongst the list of all animes
    - Popularity: popularity of the anime based on the users
    - Members: count of members of the anime
    - Favourites: count of members who have this anime as one of their favourites
    - Background_music : background_music used in the anime
    - Premiered: for a anime movie/series, date when it was premiered live.
    - Broadcast: was the anime broadcasted
    - Related_titles: other titles related to current anime title
    - Opening_theme: the opening theme song of the anime
    - Ending_theme: the ending theme song of the anime
    - Score: the current score of the anime
    - Num_votes: number of votes given to the anime
    - Genre: list of genres of current title
    - Studio: name of the studio producing the anime
    - Licensor: name of the licensor for the anime
    - Producer: name of the producer for the anime
    - Users: list of users id that are associated with current anime title
  - Users collection :
    - _id: Unique user id associated with a user
    - Username: username of the user on myanimelist
    - Gender: gender of the user
    - Birthdate: date of birth of the user
    - Current_watch_anime : count of animes currently watched by the user
    - Completed_anime : count of animes completed by the user
    - Anime_onhold: count of animes that are put on hold by the user
    - Dropped_anime: count of animes dropped by the user
    - Anime_plannedtowatch: count of animes that user plans to watch
    - days_spent _on_anime: count of days the user has spent on watching the animes
    - Access_rank : current access rank of the user
    - Site_join_date: date when the user joined myanimelist
    - Mean_score: average rating of all the animes rated by the user
    - Rewatched_stats: count of animes he has rewatched
    - Episodes_stats: count of episodes he has watched

- The Anime collection will act as the main body of the database and the users collection is the supporting collection.

The steps to load the anime and users data are as follows:

- **Step 1:** Export the current data stored in the postgresql to a json file. This step copies all the data from the postgres data to the json format.

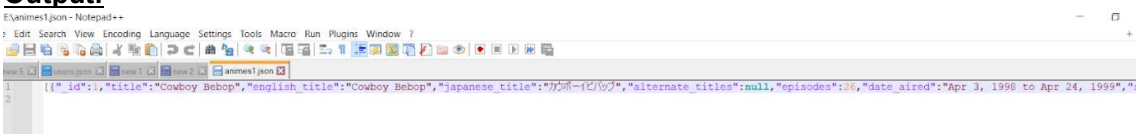
Expected Output : The output to the export would a single json array that will have multiple json objects, where in the each json object will signify the row from the postgres table.
(Note – script exportToMongo.sql can be used to review the code for this section)

- Anime Collection:

Query:

```
COPY(SELECT json_agg(row_to_json(anime)) from (SELECT a.anime_id as
"_id",a.title as title, a.title_english as english_title,
a.title_japanese as japanese_title,a.title_synonyms as alternate_titles,
a.title_episodes as episodes, a.aired as date_aired,
a.aired_from_to as start_end_date, a.duration as duration,
a.is_airing as currently_airing, a.rank as rank,
a.popularity as popularity, a.members as members,
a.favourites as favourites, a.background as background_music,
a.premiered as premiered, a.broadcast as broadcast,
a.related_to as related_titles, a.opening_theme as opening_theme,
a.ending_theme as ending_theme, a.score as score,
a.num_votes as num_votes,
string_agg(DISTINCT g.genre, ',') as genre,
      string_agg(DISTINCT s.studio , ',') as studio,
      string_agg(DISTINCT l.licensor, ',') as licensor,
      string_agg(DISTINCT p.producer, ',') as producer,
      string_agg(DISTINCT CAST(w.user_id as text), ',') as users
   from anime a
   inner join has_genre hg on a.anime_id = hg.anime_id
   inner join genre g on hg.genre_id = g.genre_id
   left join created_by cb on a.anime_id = cb.anime_id
   left join studio s on cb.studio_id = s.studio_id
   left join licensed_by lb on a.anime_id = lb.anime_id
   left join licensor l on lb.licensor_id = l.licensor_id
   left join produced_by pb on a.anime_id = pb.anime_id
   left join producer p on pb.producer_id = p.producer_id
   left join watches w on a.anime_id = w.anime_id
   left join users u on w.user_id = u.user_id
group by a.anime_id order by a.anime_id) as anime) TO 'E:\animes.json';
```

Output:



Execution Time:

```
completed in 5 h 35 m 48 s 332 ms
```

- Users Collection :

Query:

```
COPY (SELECT json_agg(row_to_json(new_user)) from
   (SELECT Users.user_id as "_id", Users.username as username , Users.gender
    as gender,
        Users.birth as "birthdate", Users.user_watching as "current_watch_anime",
        Users.user_completed as "completed_anime", Users.user_onhold as
    "anime_onhold",
        Users.user_dropped as "dropped_anime", Users.user_plantowatch as
    "anime_plannedtowacth",
        Users.user_days_spent_watching as "days_spent_on_anime",
    Users.access_rank as "access_rank",
        Users.join_date as "site_join_date", Users.stats_mean_score as
    "mean_score",
        Users.stats_rewatched as "rewatched_stats", Users.stats_episodes as
    "episodes_stats"
   FROM Users) as new_user) TO 'E:\users.json';
```

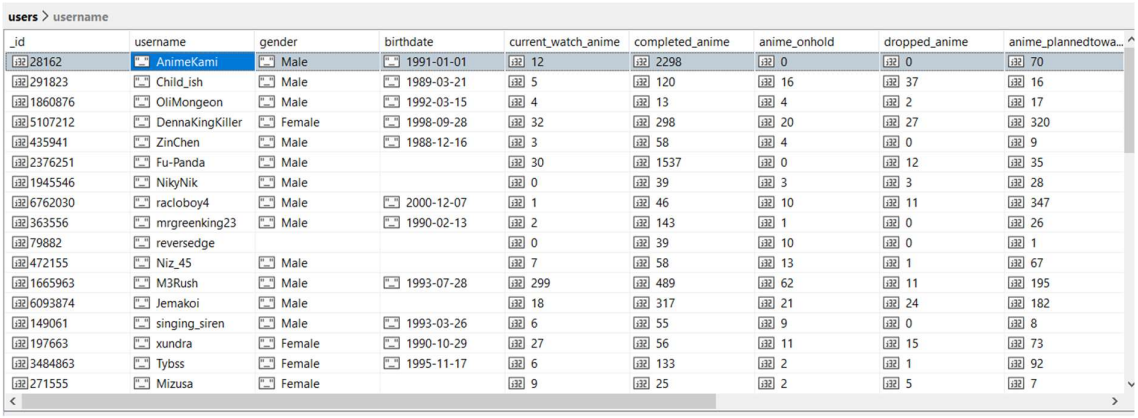
Output:

Execution Time:

```
anime> COPY (SELECT json_agg(row_to_json(new_user)) from
          (SELECT Users.user_id as "_id", Users.username as username , Users.gender as gender,
                  Users.birth as "birthdate", Users.user_watching as "current_watch_anime",
                  Users.user_completed as "completed_anime", Users.user_onhold as "anime_onhold",
                  Users.user_dropped as "dropped_anime", Users.user_plantowatch as "anime_plannedtowacth",
                  Users.user_days_spent_watching as "days_spent_on_anime", Users.access_rank as "access_rank",
                  Users.join_date as "site_join_date", Users.stats_mean_score as "mean_score",
                  Users.stats_rewatched as "rewatched_stats", Users.stats_episodes as "episodes_stats"
           FROM Users) as new_user) TO 'E:\users.json'
[2023-03-30 22:29:05] 1 row affected in 18 s 327 ms
```

- **Step 2:** After exporting the data to a json file, use either a python script to load the data into MongoDB. While loading the data through the python program, we handle the data cleaning operations.
  (Note – Script mongodb_datainsert.py can be used to review the code in this section)

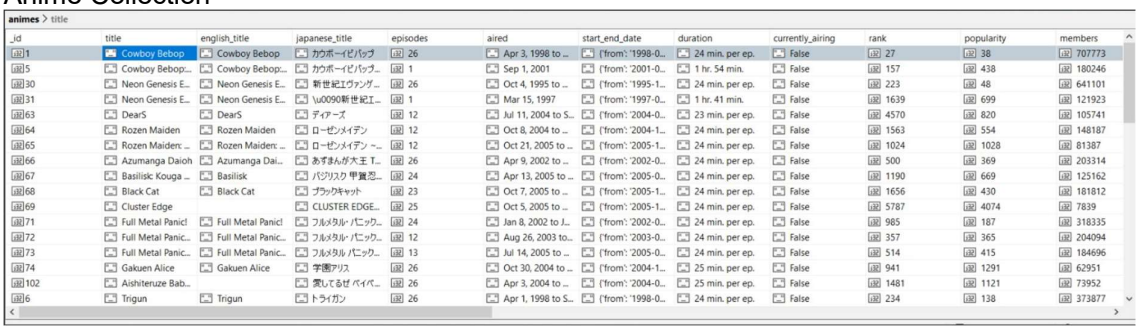
  User Collection –



users > username

_id	username	gender	birthdate	current_watch_anime	completed_anime	anime_onhold	dropped_anime	anime_plannedtowa...
28162	AnimeKami	Male	1991-01-01	12	2298	0	0	70
291823	Child_ish	Male	1989-03-21	5	120	16	37	16
1860876	OliMongeon	Male	1992-03-15	4	13	4	2	17
5107212	DennaKingKiller	Female	1998-09-28	32	298	20	27	320
435941	ZinChen	Male	1988-12-16	3	58	4	0	9
2376251	Fu-Panda	Male		30	1537	0	12	35
1945546	NikyNik	Male		0	39	3	3	28
6762030	racloboy4	Male	2000-12-07	1	46	10	11	347
363556	mrgreenking23	Male	1990-02-13	2	143	1	0	26
79882	reversedge			0	39	10	0	1
472155	Niz_45	Male		7	58	13	1	67
1665963	M3Rush	Male	1993-07-28	299	489	62	11	195
6093874	Jemakoi	Male		18	317	21	24	182
149061	singing_siren	Male	1993-03-26	6	55	9	0	8
197663	xundra	Female	1990-10-29	27	56	11	15	73
3484863	Tybss	Female	1995-11-17	6	133	2	1	92
271555	Mizusa	Female		9	25	2	5	7

  Anime Collection –



animes > title

_id	title	english_title	japanese_title	episodes	aired	start_end_date	duration	currently_airing	rank	popularity	members
1	Cowboy Bebop	Cowboy Bebop	カウボーイビバップ	26	Apr 3, 1998 to ...	{'from': '1998-0...	24 min. per ep.	False	27	38	707773
5	Cowboy Bebop:...	Cowboy Bebop:...	カウボーイビバップ...	1	Sep 1, 2001	{'from': '2001-0...	1 hr. 54 min.	False	157	438	180246
30	Neon Genesis E...	Neon Genesis E...	新世紀エヴァンゲ...	26	Oct 4, 1995 to ...	{'from': '1995-1...	24 min. per ep.	False	223	48	641101
31	Neon Genesis E...	Neon Genesis E...	\u0090新世紀エ...	1	Mar 15, 1997	{'from': '1997-0...	1 hr. 41 min.	False	1639	699	121923
63	DearS	DearS	ディアーズ	12	Jul 11, 2004 to S...	{'from': '2004-0...	23 min. per ep.	False	4570	820	105741
64	Rozen Maiden	Rozen Maiden	ローゼンメイデン	12	Oct 8, 2004 to ...	{'from': '2004-1...	24 min. per ep.	False	1563	554	148187
65	Rozen Maiden: ...	Rozen Maiden: ...	ローゼンメイデン ~...	12	Oct 21, 2005 to ...	{'from': '2005-1...	24 min. per ep.	False	1024	1028	81387
66	Azumanga Daioh	Azumanga Dai...	あずまんが大王 T...	26	Apr 9, 2002 to ...	{'from': '2002-0...	24 min. per ep.	False	500	369	203314
67	Basilisk: Kouga ...	Basilisk	バジリスク 甲賀忍...	24	Apr 13, 2005 to ...	{'from': '2005-0...	24 min. per ep.	False	1190	669	125162
68	Black Cat	Black Cat	ブラックキャット	23	Oct 7, 2005 to ...	{'from': '2005-1...	24 min. per ep.	False	1656	430	181812
69	Cluster Edge		CLUSTER EDGE...	25	Oct 5, 2005 to ...	{'from': '2005-1...	24 min. per ep.	False	5787	4074	7839
71	Full Metal Panic!	Full Metal Panic!	フルメタル パニック...	24	Jan 8, 2002 to J...	{'from': '2002-0...	24 min. per ep.	False	985	187	318335
72	Full Metal Panic...	Full Metal Panic...	フルメタル パニック...	12	Aug 26, 2003 to...	{'from': '2003-0...	24 min. per ep.	False	357	365	204094
73	Full Metal Panic...	Full Metal Panic...	フルメタル パニック...	13	Jul 14, 2005 to ...	{'from': '2005-0...	24 min. per ep.	False	514	415	184696
74	Gakuen Alice	Gakuen Alice	学園アリス	26	Oct 30, 2004 to ...	{'from': '2004-1...	25 min. per ep.	False	941	1291	62951
102	Aishiteruze Bab...		愛してるぜ ベイベ...	26	Apr 3, 2004 to ...	{'from': '2004-0...	25 min. per ep.	False	1481	1121	73952
6	Trigun	Trigun	トライガン	26	Apr 1, 1998 to S...	{'from': '1998-0...	24 min. per ep.	False	234	138	373877

a. How is invalid data, that is null/ incorrect values are handled?

- As we are exporting the data from postgres in order to import it into mongo we can be sure that there would be no incorrect data format / value inserted.
- The null values are explicitly handled through the python script before inserting to the mongo collection.

b. How is it different from relational databases?

- In a document-oriented database, the database gives a higher level of control over the individual attributes of the row as there are independent individual entities with similar structure of the data.
- We can have rows where the same attributes can have different formats. When validation is handled before the data is inserted, then mongo dB insertion is relatively lesser constraint - restricted.
- Mongo is much more scalable than postgres. We do not need to concerned about attribute level validation when new data is inserted.

2. Queries over Postgres Anime database.

(**Note –** File interestingQueries.sql can be used to refer to the code pertaining to this section.)

1. How many drama anime has Animax produced?
   1. Query -

```
SELECT a.anime_id, g.genre, p.producer
FROM anime a
    INNER JOIN has_genre hg ON a.anime_id = hg.anime_id
    INNER JOIN genre g ON hg.genre_id = g.genre_id
    INNER JOIN produced_by pb ON a.anime_id = pb.anime_id
    INNER JOIN producer p ON p.producer_id = pb.producer_id
AND p.producer = 'Animax'
AND g.genre = 'Drama';
```

   b. Execution Time -

10 rows retrieved starting from 1 in 83 ms (execution: 42 ms, fetching: 41 ms)

```
anime.public> SELECT a.anime_id, g.genre, p.producer
              FROM anime a
                  INNER JOIN has_genre hg ON a.anime_id = hg.anime_id
                  INNER JOIN genre g ON hg.genre_id = g.genre_id
                  INNER JOIN produced_by pb ON a.anime_id = pb.anime_id
                  INNER JOIN producer p ON p.producer_id = pb.producer_id
              AND p.producer = 'Animax'
              AND g.genre = 'Drama'
[2023-03-30 11:15:56] 10 rows retrieved starting from 1 in 83 ms (execution: 42 ms, fetching: 41 ms)
```

   c. Proof of Data -

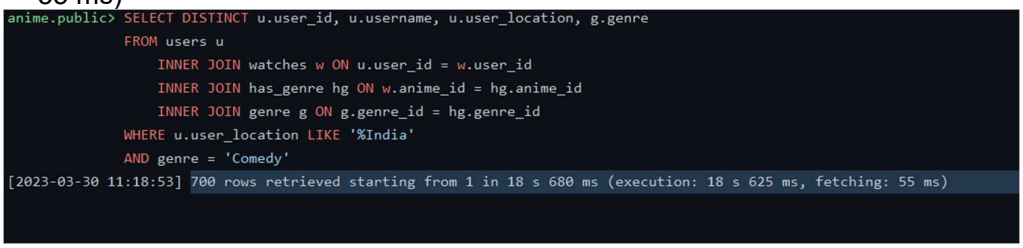
	anime_id	genre	producer
1	16067	Drama	Animax
2	1520	Drama	Animax
3	990	Drama	Animax
4	170	Drama	Animax
5	1377	Drama	Animax
6	102	Drama	Animax
7	4884	Drama	Animax
8	16201	Drama	Animax
9	15059	Drama	Animax
10	5200	Drama	Animax

2. Which users have been watching comedy genre anime in India ?
   1. Query -

```
SELECT DISTINCT u.user_id, u.username, u.user_location, g.genre
FROM users u
    INNER JOIN watches w ON u.user_id = w.user_id
    INNER JOIN has_genre hg ON w.anime_id = hg.anime_id
    INNER JOIN genre g ON g.genre_id = hg.genre_id
WHERE u.user_location LIKE '%India'
AND genre = 'Comedy';
```

   b. Execution Time -

700 rows retrieved starting from 1 in 18 s 680 ms (execution: 18 s 625 ms, fetching: 55 ms)



   c. Proof of Data -

user_id	username	user_location	genre
5718	abhin4v	India	Comedy
11443	mack123	Thane, MH, India	Comedy
28630	herlepras	Bangalore,India	Comedy
30983	sharathpuranik	Bangalore, India	Comedy
40362	viveckvivu	India	Comedy
57726	stupidsama	Kolkata,West Bengal,India	Comedy
67833	alienninjasaiyan	New Delhi, India	Comedy
76277	dPsychc	Kolkata, West Bengal, India	Comedy
78218	vinumsv	Chennai,India	Comedy
79230	an_ink_pen	India	Comedy
81809	Adipvpster	Karnataka, India	Comedy
109528	Sparx75	Kolkata, India	Comedy
115174	Ne0	India	Comedy
123201	chronodekar	Kochi,India	Comedy

3. Which licensor licensed the most anime post 2000s ?
   1. Query -

```
SELECT licensor.licensor_id AS ID, licensor.licensor, count(licensed_by.anime_id)
FROM licensor
    INNER JOIN licensed_by ON licensor.licensor_id = licensed_by.licensor_id
    INNER JOIN anime ON licensed_by.anime_id = anime.anime_id
WHERE cast(substr(aired_from_to,11,4) AS INT) > 2000 AND aired_from_to NOT
    LIKE '%None%'
GROUP BY licensor.licensor_id, licensor.licensor ORDER BY
    count(licensed_by.anime_id) DESC ;
```

   b. Execution Time -
   63 rows retrieved starting from 1 in 57 ms (execution: 19 ms, fetching: 38 ms)

```
anime.public> SELECT licensor.licensor_id AS ID, licensor.licensor, count(licensed_by.anime_id)
              FROM licensor
                  INNER JOIN licensed_by ON licensor.licensor_id = licensed_by.licensor_id
                  INNER JOIN anime ON licensed_by.anime_id = anime.anime_id
              WHERE cast(substr(aired_from_to,11,4) AS INT) > 2000 AND aired_from_to NOT LIKE '%None%'
              GROUP BY licensor.licensor_id, licensor.licensor ORDER BY count(licensed_by.anime_id) DESC
[2023-03-30 11:34:23] 63 rows retrieved starting from 1 in 57 ms (execution: 19 ms, fetching: 38 ms)
```

   c. Proof of Data -

id	licensor	count
34	Funimation	818
58	Sentai Filmworks	605
1	Media Blasters	158
12	Aniplex of America	138
63	Bandai Entertainment	133
16	ADV Films	133
17	Viz Media	124
21	Geneon Entertainment USA	104
36	Discotek Media	81
45	Nozomi Entertainment	67
30	Inc.	51
67	NIS America	51
61	NYAV Post	46

4. Which studio has produced the most 9+ rated animes produced by a studio ?
   1. Query -

```
SELECT studio.studio, count(anime.score)
FROM created_by INNER JOIN studio ON created_by.studio_id = studio.studio_id
                INNER JOIN anime ON created_by.anime_id = anime.anime_id
WHERE score > 9.0 GROUP BY studio.studio ORDER BY count(anime.score)
   DESC;
```

b. Execution Time -
11 rows retrieved starting from 1 in 44 ms (execution: 18 ms, fetching: 26 ms)

```
anime.public> SELECT studio.studio, count(anime.score)
              FROM created_by INNER JOIN studio ON created_by.studio_id = studio.studio_id
                              INNER JOIN anime ON created_by.anime_id = anime.anime_id
              WHERE score > 9.0 GROUP BY studio.studio ORDER BY count(anime.score) DESC
[2023-03-30 11:35:39] 11 rows retrieved starting from 1 in 44 ms (execution: 18 ms, fetching: 26 ms)
```

c. Proof of Data -

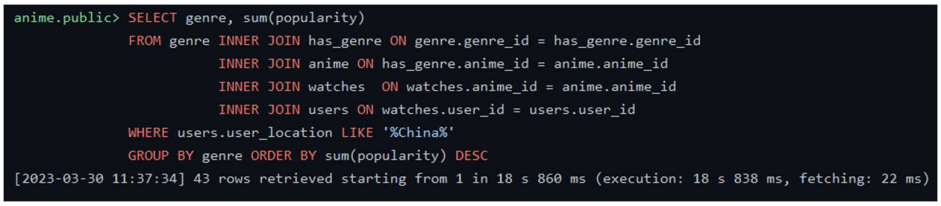
	studio	count
1	Sunrise	4
2	Bandai Namco Pictures	2
3	White Fox	2
4	Kyoto Animation	2
5	Shaft	2
6	Bones	1
7	Madhouse	1
8	Magic Bus	1
9	Artland	1
10	Collaboration Works	1
11	CoMix Wave Films	1

5. Which are the most popular genres across all anime in China?
   1. Query -

```
SELECT genre, sum(popularity)
FROM genre INNER JOIN has_genre ON genre.genre_id = has_genre.genre_id
        INNER JOIN anime ON has_genre.anime_id = anime.anime_id
        INNER JOIN watches  ON watches.anime_id = anime.anime_id
        INNER JOIN users ON watches.user_id = users.user_id
WHERE users.user_location LIKE '%China%'
GROUP BY genre ORDER BY sum(popularity) DESC;
```

   b. Execution Time -

43 rows retrieved starting from 1 in 18 s 860 ms (execution: 18 s 838 ms, fetching: 22 ms



   c. Proof of Data -

	genre	sum
1	Comedy	22825699
2	Action	15303176
3	Sci-Fi	10719802
4	Drama	10678798
5	Fantasy	10141791
6	Romance	9712596
7	Adventure	9523567
8	Shounen	8948637
9	School	8189144
10	Slice of Life	7211524
11	Supernatural	6430069
12	Mecha	5039966

Possible Indexing -
Note - createIndex.sql can be used to review code for this section.

Indexes were added on the following columns:

- score in anime table
- popularity in anime table
- user_location in users table
- Aired_from_to in the anime table

Execution Time	Without Indexes	With Indexes
Query 1	83 ms	49 ms
Query 2	18 s 680 ms	18 s 102 ms
Query 3	57 ms	43 ms
Query 4	44 ms	35 ms
Query 5	18 s 860 ms	17 s 408 ms

The table above shows a comparison of the query execution time before and after adding indexes. Indexes act as pointers to data in our tables that allow for a faster retrieval. It was chosen to add indexes to the columns above because they were most frequently used in our queries excluding columns that were already declared as primary keys. Primary keys are already indexed so there was no need to add an additional index on those columns.

The proof of the decreased execution time is below in order of Query 1 to Query 5:

```
anime.public> SELECT a.anime_id, g.genre, p.producer
              FROM anime a
                  INNER JOIN has_genre hg ON a.anime_id = hg.anime_id
                  INNER JOIN genre g ON hg.genre_id = g.genre_id
                  INNER JOIN produced_by pb ON a.anime_id = pb.anime_id
                  INNER JOIN producer p ON p.producer_id = pb.producer_id
              AND p.producer = 'Animax'
              AND g.genre = 'Drama'
[2023-03-30 12:26:12] 10 rows retrieved starting from 1 in 49 ms (execution: 14 ms, fetching: 35 ms)
anime.public> SELECT DISTINCT u.user_id, u.username, u.user_location, g.genre
              FROM users u
                  INNER JOIN watches w ON u.user_id = w.user_id
                  INNER JOIN has_genre hg ON w.anime_id = hg.anime_id
                  INNER JOIN genre g ON g.genre_id = hg.genre_id
              WHERE u.user_location LIKE '%India'
              AND genre = 'Comedy'
[2023-03-30 12:28:55] 700 rows retrieved starting from 1 in 18 s 102 ms (execution: 18 s 55 ms, fetching: 47 ms)
anime.public> SELECT licensor.licensor_id AS ID, licensor.licensor, count(licensed_by.anime_id)
              FROM licensor
                  INNER JOIN licensed_by ON licensor.licensor_id = licensed_by.licensor_id
                  INNER JOIN anime ON licensed_by.anime_id = anime.anime_id
              WHERE cast(substr(aired_from_to,11,4) AS INT) > 2000 AND aired_from_to NOT LIKE '%None%'
              GROUP BY licensor.licensor_id, licensor.licensor ORDER BY count(licensed_by.anime_id) DESC
[2023-03-30 12:30:26] 63 rows retrieved starting from 1 in 43 ms (execution: 17 ms, fetching: 26 ms)
anime.public> SELECT studio.studio, count(anime.score)
              FROM created_by INNER JOIN studio ON created_by.studio_id = studio.studio_id
                              INNER JOIN anime ON created_by.anime_id = anime.anime_id
              WHERE score > 9.0 GROUP BY studio.studio ORDER BY count(anime.score) DESC
[2023-03-30 12:30:55] 11 rows retrieved starting from 1 in 35 ms (execution: 11 ms, fetching: 24 ms)
anime.public> SELECT genre, sum(popularity)
              FROM genre INNER JOIN has_genre ON genre.genre_id = has_genre.genre_id
                         INNER JOIN anime ON has_genre.anime_id = anime.anime_id
                         INNER JOIN watches  ON watches.anime_id = anime.anime_id
                         INNER JOIN users ON watches.user_id = users.user_id
              WHERE users.user_location LIKE '%China%'
              GROUP BY genre ORDER BY sum(popularity) DESC
[2023-03-30 12:31:33] 43 rows retrieved starting from 1 in 17 s 408 ms (execution: 17 s 379 ms, fetching: 29 ms)
```

3. Functional dependencies and normalization

- **Functional Dependencies**

1. anime_id -> title, title_english, title_japanese, title_synonyms, aired, aired_from_to, duration, isAiring, rank, popularity, members, favorites, background, premiered, broadcast, related_to, opening_theme, ending_theme, score, num_votes
    a. The anime id is the primary key for the Anime table. Each anime id uniquely identifies an anime, so all other attributes in the table depend on it.
    The anime id uniquely determines a unique set: title, title in English, title in Japanese, synonyms of title, duration, airing date(aired_from_to) and current airing status, rank, popularity, number of members, favorites, background, premiered season, broadcast time, related_to, the opening theme, the ending theme, score, and number of votes.

2. user_id -> username, gender, birth, user_watching, user_completed, user_onhold, user_dropped, user_plantowatch, user_days_spent_watching, user_location, access_rank, join_date, stats_mean_score, stats_rewatched, stats_episode
    a. The user id is the primary key for the Users table. Each user id uniquely identifies a user, so all other attributes in the table depend on it.
    Each user id maps to a unique set of username, gender, birth, what a user is watching, what a user has completed, what a user has paused(user_onhold), what a user lefts midway(user_dropped), what a user plans to watch(user_plantowatch), the days a user spent on watching, user's location, access rank, join date, mean score, rewatched times, and the number of the watched episode.

3. genre_id -> genre
    a. The genre id is the primary key for the Genre table. Each genre id uniquely identifies a genre, so the genre attribute depends on it.
    Given that the genre id is a unique int value for a genre, the genre id uniquely identifies the genre.

4. producer_id -> producer
    a. The producer id is the primary key for the Producer table. Each producer id uniquely identifies a producer, so the producer attribute depends on it.
    Given that the producer id is a unique int value for a genre, the producer id uniquely identifies the producer.

5. licensor_id -> licensor
    a. The licensor id is the primary key for the Licensor table. Each licensor id uniquely identifies a licensor, so the licensor attribute depends on it.
    Given that the licensor id is a unique int value for a licensor, the licensor id uniquely identifies the licensor.

6. studio_id -> studio
    a. The studio id is the primary key for the Studio table. Each studio id uniquely identifies a studio, so the studio attribute depends on it.
    Given that the studio id is a unique int value for a studio, the studio id uniquely identifies the studio.

7. user_id, anime_id -> my_watched_episodes, my_start_date, my_finish_date, my_score, my_status, my_rewatching, my_rewatching_ep, my_last_updated, my_tags

a. The primary key for the Watches table is the combination of the user id and anime id. It means that each combination of user_id and anime_id uniquely identifies a specific user's watch history for a particular anime. All other attributes in the table depend on this combination.
   Given the user id and anime id, a unique tuple consisting of the user's watched episodes, the start date of watching, finish date of watching, the user's score, the user's watching status, the user's rewatching, the user's rewatching episode, the user's last updated, and the user's tags can be retrieved.

- **Data Normalization**

  First Normal Form (1NF):
  - All attributes in a table should be atomic, meaning they cannot be further divided into smaller parts. And there are no repeating groups. This condition is already satisfied in all tables.

  Second Normal Form (2NF):
  - A table is in 2NF if it is in 1NF and all non-key attributes are fully dependent on the primary key. All tables are in 2NF as there are no partial dependencies. For example, in the Watches table, all non-key attributes depend on the entire primary key (user_id, anime_id) and not on a part of it.

  Third Normal Form (3NF):
  - A table is in 3NF if it is in 2NF and has no transitive dependencies.
  - All tables are in 3NF as there are no transitive dependencies. For example, in the Anime table, all non-key attributes depend directly on the primary key anime_id and not on any other non-key attribute.

  Therefore, all the tables in Phase I already follow normalization rules up to 3NF, minimizing data redundancy and ensuring data integrity.