IMDb Top 1000 Movies Analysis

Mengyan Wu

I Data Source

I used *The IMDb Top 1000 Movies from 2006 to 2016* dataset from Kaggle for this project.
https://www.kaggle.com/PromptCloudHQ/imdb-data.
The dataset contains data of the most popular movies on IMDb released from 2006 to 2016. The data fields include Title, Genre, Description, Director, Actors, Year, Runtime and Rating.

Sample Data:

	Rank	Title	Genre	Description	Director	Actors	Year	Runtime (Minutes)	Rating
0	1	Guardians of the Galaxy	Action,Adventure,Sci-Fi	A group of intergalactic criminals are forced ...	James Gunn	Chris Pratt, Vin Diesel, Bradley Cooper, Zoe S...	2014	121	8.1
1	2	Prometheus	Adventure,Mystery,Sci-Fi	Following clues to the origin of mankind, a te...	Ridley Scott	Noomi Rapace, Logan Marshall-Green, Michael Fa...	2012	124	7.0
2	3	Split	Horror,Thriller	Three girls are kidnapped by a man with a diag...	M. Night Shyamalan	James McAvoy, Anya Taylor-Joy, Haley Lu Richar...	2016	117	7.3
3	4	Sing	Animation,Comedy,Family	In a city of humanoid animals, a hustling thea...	Christophe Lourdelet	Matthew McConaughey,Reese Witherspoon, Seth Ma...	2016	108	7.2
4	5	Suicide Squad	Action,Adventure,Fantasy	A secret government agency recruits some of th...	David Ayer	Will Smith, Jared Leto, Margot Robbie, Viola D...	2016	123	6.2
...	...	...	...	...	...	...	...	...	...

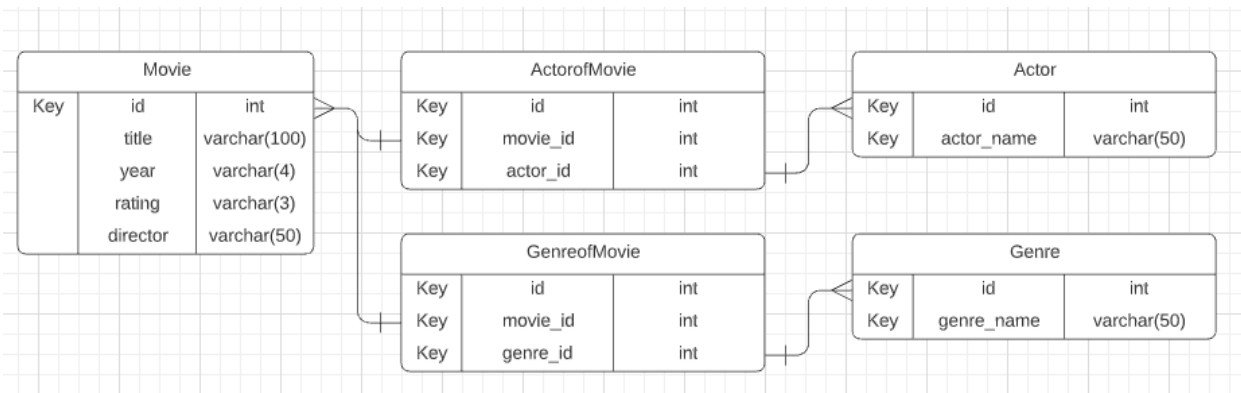
II Design of Database

The raw dataset have 1000 rows. The dataset need to be normalized to eliminate redundancy and facilitate consistent modification of data.
The raw dataset have the following problems to consider in the normalization:
1. each movie have a set of actors which are co-listed in the “Actors” field.
2. each movie have a set of genres which are co-listed in the “Genre” field.
This leads to redundancy and inconsistency.

Thus, the data need to be re-organized:
1. For each movie, the original set of actors should be exploded to multiple rows of data, with each row containing only one actor name and the same information of that movie. (I followed the instruction of the “Data Normalization” page on Canvas)

2. There should be a Movie table containing the information of each movie, an Actor table containing the id of each actor and the name of the actor, and a Genre table containing the id of each genre and the name of the genre.
3. There should also be an ActorofMovie table which linked the Movie table and the Actor table by representing the actor name by the id of the actor and the movie by the id of the movie. Also, there should also be a GenreofMovie table which linked the Movie table and the Genre table by representing the genre name by the id of the genre and the movie by the id of the movie. Then, the Movie and Actor/Genre datasets are linked with key (id and movie_id/actor_id/genre_id) instead of strings.
4. Also, since this is the top 1000 movies dataset, the "Rank" field is naturally uniquely assigned to each movie, starting from 1 and ending with 1000. So, I used Rank directly as the primary key for simplicity.

The ERD diagram of the designed database is shown below:



II Data Cleaning & Re-organization

I used pandas to do this job. The detailed code is stored in the "mengyanw_si564_project.ipynb" file.

The normalized tables I got are displayed below:

Movie:

	Rank	Title	Year	Rating	Director
0	1	Guardians of the Galaxy	2014	8.1	James Gunn
1	2	Prometheus	2012	7.0	Ridley Scott
2	3	Split	2016	7.3	M. Night Shyamalan
3	4	Sing	2016	7.2	Christophe Lourdelet
4	5	Suicide Squad	2016	6.2	David Ayer
...	...	...	...	...	...
995	996	Secret in Their Eyes	2015	6.2	Billy Ray
996	997	Hostel: Part II	2007	5.5	Eli Roth
997	998	Step Up 2: The Streets	2008	6.2	Jon M. Chu
998	999	Search Party	2014	5.6	Scot Armstrong
999	1000	Nine Lives	2016	5.3	Barry Sonnenfeld

1000 rows × 5 columns

Genre:

	genre_id	Genre
0	1	Action
1	2	Adventure
2	3	Sci-Fi
3	4	Mystery
4	5	Horror
5	6	Thriller
6	7	Animation
7	8	Comedy
8	9	Family
9	10	Fantasy
10	11	Drama

GenreofMovie:

	movie_id	genre_id
0	1	1
1	1	2
2	1	3
3	2	2
4	2	4
...	...	...
2550	999	2
2551	999	8
2552	1000	8
2553	1000	9
2554	1000	10

2555 rows × 2 columns

Actor:

	actor_id	Actor
0	1	Chris Pratt
1	2	Vin Diesel
2	3	Bradley Cooper
3	4	Zoe Saldana
4	5	Noomi Rapace
...	...	...
2389	2390	Adam Pally
2390	2391	Thomas Middleditch
2391	2392	Shannon Woodward
2392	2393	Kevin Spacey
2393	2394	Cheryl Hines

2394 rows × 2 columns

ActorofMovie:

	movie_id	actor_id
0	1	1
1	1	2
2	1	3
3	1	4
4	2	5
...	...	...
3994	999	2392
3995	1000	2393
3996	1000	254
3997	1000	1731
3998	1000	2394

3999 rows × 2 columns

IV Create Database & Insert Data

Then I created a MySQL database named imdb and insert the normalized tables through DataGrip.

1. Create Database

```
[2021-12-04 22:43:05] Connected
> create schema imdb
[2021-12-04 22:43:06] completed in 358 ms
> use imdb
[2021-12-04 22:45:47] completed in 84 ms
```

2. Create Tables & Insert Data from .csv files

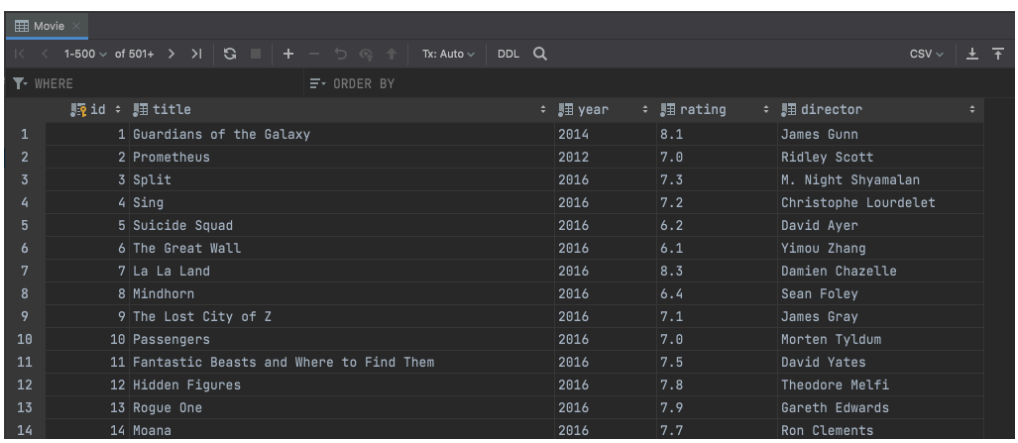
<u>*Movie:*</u>

```
imdb> create table Movie
    (
        id int auto_increment,
        title varchar(100) not null,
        year varchar(4) not null,
        rating varchar(3) not null,
        director varchar(50) not null,
        constraint Movie_pk
            primary key (id)
    )
[2021-12-04 22:52:13] completed in 97 ms
```

***I used the "import data from file" option in Datagrip to insert data*

22:54 class_server

movies.csv imported to Movie: 1,000 rows (56 sec, 553 ms, 837 B/s)



Movie

1-500 of 501+ | Tx: Auto | DDL | CSV

WHERE | ORDER BY

	id	title	year	rating	director
1	1	Guardians of the Galaxy	2014	8.1	James Gunn
2	2	Prometheus	2012	7.0	Ridley Scott
3	3	Split	2016	7.3	M. Night Shyamalan
4	4	Sing	2016	7.2	Christophe Lourdelet
5	5	Suicide Squad	2016	6.2	David Ayer
6	6	The Great Wall	2016	6.1	Yimou Zhang
7	7	La La Land	2016	8.3	Damien Chazelle
8	8	Mindhorn	2016	6.4	Sean Foley
9	9	The Lost City of Z	2016	7.1	James Gray
10	10	Passengers	2016	7.0	Morten Tyldum
11	11	Fantastic Beasts and Where to Find Them	2016	7.5	David Yates
12	12	Hidden Figures	2016	7.8	Theodore Melfi
13	13	Rogue One	2016	7.9	Gareth Edwards
14	14	Moana	2016	7.7	Ron Clements

In total, there are 1000 rows in the Movie table.

<u>*Actor:*</u>

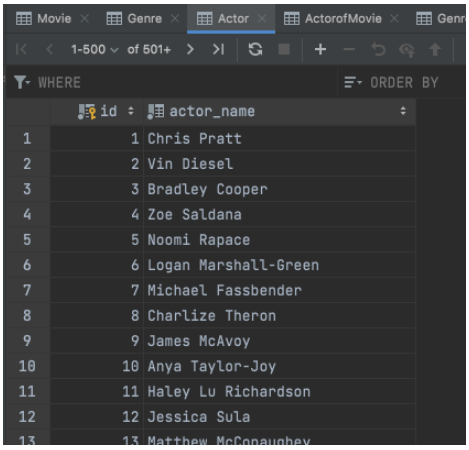
```
imdb> create table Actor
    (
        id int auto_increment,
```

```
        actor_name varchar(50) not null,
        constraint Actor_pk
                primary key (id)
    )
```

[2021-12-04 23:11:21] completed in 118 ms

***I used the "import data from file" option in Datagrip to insert data*

23:25 class_server: actor.csv imported to Actor: 2,394 rows (381 ms)



	id	actor_name
1	1	Chris Pratt
2	2	Vin Diesel
3	3	Bradley Cooper
4	4	Zoe Saldana
5	5	Noomi Rapace
6	6	Logan Marshall-Green
7	7	Michael Fassbender
8	8	Charlize Theron
9	9	James McAvoy
10	10	Anya Taylor-Joy
11	11	Haley Lu Richardson
12	12	Jessica Sula
13	13	Matthew McConaughey

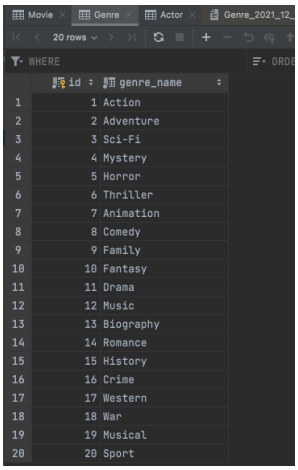
<u>Genre:</u>

```
imdb> create table Genre
    (
        id int auto_increment,
        genre_name varchar(50) not null,
        constraint Genre_pk
                primary key (id)
    )
```

[2021-12-04 23:22:41] completed in 78 ms

***I used the "import data from file" option in Datagrip to insert data*

23:23 class_server: genre.csv imported to Genre: 20 rows (51 ms)



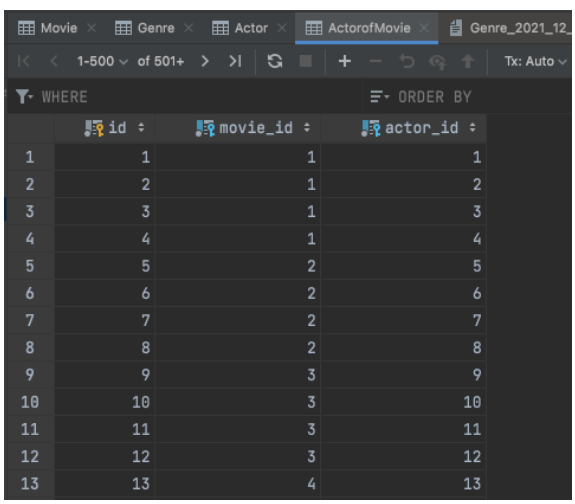
	id	genre_name
1	1	Action
2	2	Adventure
3	3	Sci-Fi
4	4	Mystery
5	5	Horror
6	6	Thriller
7	7	Animation
8	8	Comedy
9	9	Family
10	10	Fantasy
11	11	Drama
12	12	Music
13	13	Biography
14	14	Romance
15	15	History
16	16	Crime
17	17	Western
18	18	War
19	19	Musical
20	20	Sport

In total, there are 20 rows in the Genre table.

ActorofMovie:

```
imdb> create table ActorofMovie
    (
        id int auto_increment,
        movie_id int not null,
        actor_id int not null,
        constraint ActorofMovie_pk
            primary key (id),
        constraint ActorofMovie_Actor_id_fk
            foreign key (actor_id) references Actor (id)
                on update cascade on delete cascade,
        constraint ActorofMovie_Movie_id_fk
            foreign key (movie_id) references Movie (id)
                on update cascade on delete cascade
    )
[2021-12-04 23:27:07] completed in 138 ms
```

***I used the "import data from file" option in Datagrip to insert data*
23:30 class_server: movie_actor.csv imported to ActorofMovie: 3,999 rows (958 ms)



	id	movie_id	actor_id
1	1	1	1
2	2	1	2
3	3	1	3
4	4	1	4
5	5	2	5
6	6	2	6
7	7	2	7
8	8	2	8
9	9	3	9
10	10	3	10
11	11	3	11
12	12	3	12
13	13	4	13

GenreofMovie:

```
imdb> create table GenreofMovie
    (
        id int auto_increment,
        movie_id int not null,
        genre_id int not null,
        constraint GenreofMovie_pk
            primary key (id),
        constraint GenreofMovie_Genre_id_fk
            foreign key (genre_id) references Genre (id)
                on update cascade on delete cascade,
```

```
        constraint GenreofMovie_Movie_id_fk
                foreign key (movie_id) references Movie (id)
                        on update cascade on delete cascade
    )
[2021-12-04 23:33:04] completed in 130 ms
```

***I used the "import data from file" option in Datagrip to insert data*
23:34 class_server: movie_genre.csv imported to GenreofMovie: 2,555 rows (425 ms)

Movie | Genre | Actor | ActorofMovie | GenreofMovie

1-500 of 501+ Tx: Auto

WHERE ORDER BY

	id	movie_id	genre_id
1	1	1	1
2	2	1	2
3	3	1	3
4	4	2	2
5	5	2	4
6	6	2	3
7	7	3	5
8	8	3	6
9	9	4	7
10	10	4	8
11	11	4	9
12	12	5	1
13	13	5	2

V Pose Questions & Answer with SQL Queries

Q1: Find top 10 genres by the number of movies in that genre.

```
mysql> SELECT g.genre_name, count(1) as num_of_movies FROM GenreofMovie gm LEFT JOIN
Genre g ON gm.genre_id = g.id LEFT JOIN Movie m ON gm.movie_id = m.id GROUP BY g.id
ORDER BY  num_of_movies DESC LIMIT 10;
+------------+---------------+
| genre_name | num_of_movies |
+------------+---------------+
| Drama      |           513 |
| Action     |           303 |
| Comedy     |           279 |
| Adventure  |           259 |
| Thriller   |           195 |
| Crime      |           150 |
| Romance    |           141 |
| Sci-Fi     |           120 |
| Horror     |           119 |
| Mystery    |           106 |
+------------+---------------+
10 rows in set (0.24 sec)
```

Q2: Find the average rating of all movies by year order by increasing year.

```
mysql> SELECT AVG(rating), year FROM Movie GROUP BY year ORDER BY year;
+--------------------+------+
| AVG(rating)        | year |
+--------------------+------+
|  7.125000000000001 | 2006 |
| 7.1339622641509415 | 2007 |
|  6.784615384615384 | 2008 |
|   6.96078431372549 | 2009 |
|  6.826666666666668 | 2010 |
|   6.83809523809524 | 2011 |
|  6.924999999999999 | 2012 |
|  6.812087912087911 | 2013 |
|  6.837755102040816 | 2014 |
|  6.602362204724411 | 2015 |
|  6.436700336700337 | 2016 |
+--------------------+------+
11 rows in set (0.05 sec)
```

Q3: Find all Thriller movies ordered by decreasing rating, then by increasing year if ratings are the same, then by title if ratings and years are the same.

```
mysql> SELECT m.title, m.rating, m.year FROM GenreofMovie gm LEFT JOIN Genre g ON
gm.genre_id = g.id LEFT JOIN Movie m ON gm.movie_id = m.id WHERE g.genre_name =
"Thriller" ORDER BY m.rating DESC, m.year, m.title LIMIT 20;
+------------------------+--------+------+
| title                  | rating | year |
+------------------------+--------+------+
| The Departed           | 8.5    | 2006 |
| The Lives of Others    | 8.5    | 2006 |
| The Dark Knight Rises  | 8.5    | 2012 |
| No Country for Old Men | 8.1    | 2007 |
| The Bourne Ultimatum   | 8.1    | 2007 |
| Shutter Island         | 8.1    | 2010 |
| Relatos salvajes       | 8.1    | 2014 |
| The Imitation Game     | 8.1    | 2014 |
| Blood Diamond          | 8.0    | 2006 |
| Casino Royale          | 8.0    | 2006 |
| District 9             | 8.0    | 2009 |
| Black Swan             | 8.0    | 2010 |
| The Revenant           | 8.0    | 2015 |
| Forushande             | 8.0    | 2016 |
| Children of Men        | 7.9    | 2006 |
| Nightcrawler           | 7.9    | 2014 |
| Contratiempo           | 7.9    | 2016 |
| Taken                  | 7.8    | 2008 |
| Män som hatar kvinnor  | 7.8    | 2009 |
| Skyfall                | 7.8    | 2012 |
+------------------------+--------+------+
20 rows in set (0.03 sec)
```

Q4: Find the top 10 actors based on average movie rating who played at least 2 movies in 2006-2016. For each actor, give their name, average rating of the movies and the number of movies they played in. Sort the result in the descending order based on average movie rating. In case of ties, sort the rows by actor name.

```
mysql> SELECT a.actor_name, AVG(m.rating) AS avg_rating, COUNT(1) AS num_of_movies
FROM ActorofMovie am LEFT JOIN Movie m ON am.movie_id = m.id LEFT JOIN Actor a ON
am.actor_id = a.id GROUP BY a.actor_name HAVING num_of_movies >= 2 ORDER BY avg_rating
DESC, a.actor_name LIMIT 10;
+-------------------+-------------------+---------------+
| actor_name        | avg_rating        | num_of_movies |
+-------------------+-------------------+---------------+
| Aoi Yuki          |               8.5 |             2 |
| Aamir Khan        | 8.475000000000001 |             4 |
| Craig Ferguson    |                 8 |             2 |
| Omar Sy           |                 8 |             2 |
| Leonardo DiCaprio | 7.969999999999999 |            10 |
| Octavia Spencer   | 7.949999999999999 |             2 |
| Heath Ledger      |               7.9 |             2 |
| Rachel House      | 7.800000000000001 |             2 |
| Freida Pinto      |               7.8 |             2 |
| Michael Gambon    |               7.8 |             2 |
+-------------------+-------------------+---------------+
10 rows in set (0.04 sec)
```

Q5: Finding pairs of actors who co-starred in at least 2 movies together. The pairs of names must be unique. This means that 'actor A, actor B' and 'actor B, actor A' are the same pair, so only one of them should appear. For each pair of actors you print out, the two actors must be ordered alphabetically. The pairs are ordered in decreasing number of movies they co-stared in.

```
mysql>
SELECT A.actor_name AS actor1, B.actor_name AS actor2, count(1) AS num_of_movies
FROM
(SELECT a.actor_name, am.movie_id FROM  ActorofMovie am LEFT JOIN Movie m ON
am.movie_id = m.id LEFT JOIN Actor a ON am.actor_id = a.id) A,
(SELECT a.actor_name, am.movie_id FROM ActorofMovie am LEFT JOIN Movie m ON
am.movie_id = m.id LEFT JOIN Actor a ON am.actor_id = a.id) B
WHERE A.movie_id = B.movie_id AND A.actor_name < B.actor_name
GROUP BY actor1, actor2
ORDER BY num_of_movies DESC LIMIT 10;

+----------------------+------------------+---------------+
| actor1               | actor2           | num_of_movies |
+----------------------+------------------+---------------+
| Kristen Stewart      | Robert Pattinson |             4 |
| Jennifer Lawrence    | Josh Hutcherson  |             4 |
| Josh Hutcherson      | Liam Hemsworth   |             4 |
| Paul Walker          | Vin Diesel       |             4 |
| Jennifer Lawrence    | Liam Hemsworth   |             4 |
| Helena Bonham Carter | Johnny Depp      |             4 |
| Adam Sandler         | Kevin James      |             4 |
| Emma Watson          | Rupert Grint     |             4 |
| Daniel Radcliffe     | Rupert Grint     |             4 |
| Daniel Radcliffe     | Emma Watson      |             4 |
+----------------------+------------------+---------------+
10 rows in set (0.15 sec)
```