CSE 4/560 Milestone 2
FilmVerse

Kajol
UB ID: kajol
kajol@buffalo.edu

Mahimitra Chirala
UB ID: mahimitr
mahimitr@buffalo.edu

Sagar Gopalasetti
UB ID: sambasai
sambasai@buffalo.edu

I. Introduction

A. Problem Statement

Finding a good movie to watch can be a tedious and time-consuming task. With so many movies available to choose from, users often spend a lot of time searching the internet for ratings and reviews. To address this issue, we propose a movie information system that provides users with a comprehensive collection of movie data, including ratings, genres, actors, release year, and other relevant information. This system will enable users to search for and discover movies that meet their specific needs and preferences quickly and easily. By leveraging the system's advanced search and filtering capabilities, users can find top-rated films, movies in their preferred genres, films in a specific language, or movies featuring a favorite actor. This system aims to simplify the movie selection process and enhance the overall viewing experience for users.

II. Target User

A. The User

The IMDb dataset caters to a diverse audience, including researchers, data analysts, and movie enthusiasts, interested in discovering and studying movie and TV data. The intended audience comprises university researchers specializing in media and communication, marketing experts analyzing customer behavior and preferences, and film studios examining market trends. The dataset provides a valuable resource for these audiences to explore and gain insights into trends, patterns, and connections in the film industry. The comprehensive data available could enhance research efforts, improve marketing strategies, and drive innovation and success in the industry.

B. The Admin

The management of the IMDb dataset requires skilled professionals such as data administrators or data engineers. These professionals are responsible for ensuring that the data is correctly structured, maintained, and preserved. They must conduct data quality checks, make regular backups, and continuously work on improving database speed to ensure efficient and accurate data retrieval. Effective management of the database ensures that users have access to up-to-date, reliable information, which is essential for researchers, analysts, and industry professionals to make informed decisions. By employing experienced data professionals, the dataset can be continuously improved and maintained to provide users with the most comprehensive and accurate movie data available.

Identify applicable funding agency here. If none, delete this.

C. Real-life scenario description

Real-world applications of the IMDb dataset extend to market research companies that aim to examine consumer preferences and behavior toward films and TV series. The dataset provides valuable insights into the most popular genres, actors and actresses that appear most frequently in popular films, and production companies that have the most commercial success. This information can be utilized to make informed decisions about movie distribution and development, as well as marketing and advertising tactics for streaming services and movie studios. The IMDb dataset is a comprehensive and reliable resource that market research companies can utilize to gain valuable insights into the film industry and make data-driven decisions.

III. Database

There are six tables in data namely:

1) Akas table
   The relation provides alternate titles for movies and TV shows in different regions and languages, which can be useful for international marketing and distribution.
2) Basics table
   The Basics table contains fundamental information about movie and TV show titles such as title, year of release, genre, movie type, etc.
3) Crew Table
   This table contains information about the individuals who contribute to a film or TV show's production, such as directors, and writers.
4) Episodes Table
   The Episodes table contains information about episodes, including the number of seasons, and episode number of a TV show.
5) Ratings Table
   The Ratings table contains information about the movie's average rating and number of votes.

IV. BCNF

All the tables are in BCNF hence we did not have to decompose them. For all the dependencies of the form X $\rightarrow$Y. X is a super key and there are no transitive dependencies in any of the tables hence, they are in BCNF. To explain this,

dependencies for each table are listed as follows.

For the akas table,
akas(title_id, ordering, title, region, language, types, attributes, is_original_title) The Functional dependencies are:
titleId, ordering → title
titleId, ordering → region
titleId, ordering → language
titleId, ordering → types
titleId, ordering → attributes
titleId, ordering → is_original_title

For the basics table,
basics (tconst, title_type, primary_title, original_title, is_adult, start_year, end_year, runtime_minutes, genres) The Functional dependencies are:
tconst → title_type
tconst → primary_title
tconst → original_title
tconst → is_adult
tconst → start_year
tconst → end_year
tconst → runtime_minutes
tconst → genres

For crew table,
crew (tconst, directors, writers) The Functional dependencies are:
tconst → directors
tconst → writers

For episode table,
episode (tconst, parent_tconst, season_number, episode_number) The Functional dependencies are:
tconst → parent_tconst
tconst → season_number
tconst → episode_number

For ratings table,
ratings (tconst, average_rating, num_votes) The Functional dependencies are:
tconst → average_rating
tconst → num_votes

All the above tables are in BCNF as, in all the FDs of the form X→ Y.

V. Entity-Relationship Diagram

ER Diagram stands for Entity Relationship Diagram, It displays the relationship of entity sets stored in a database. In other words, ER diagrams help to explain the logical structure of databases.

From Figure 1, we can observe that there are six entities, namely basics, akas, crew, episode, and ratings. All the entities are related to the basics entity.

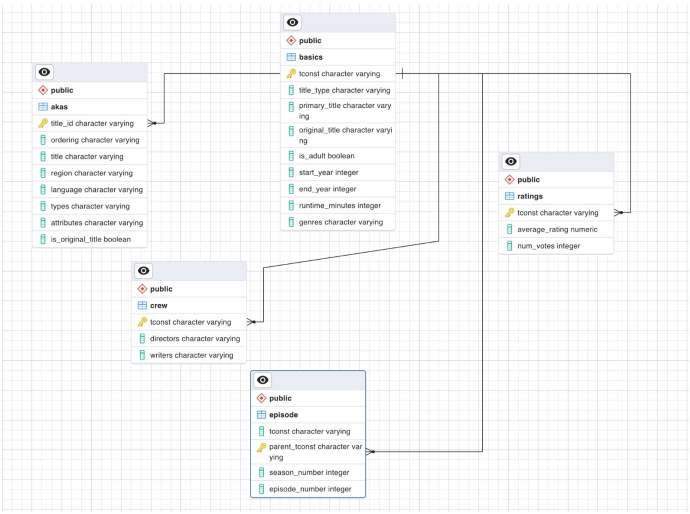


Fig. 1. ER-diagram.

VI. Constraints

The Constraints for our database are listed below as

For the akas table:
Primary Key: (title_id, ordering), The combination of these two columns uniquely identifies a row in the table, which is necessary for data integrity and avoiding duplicate entries.

Foreign Key: The foreign key is title_id, which refers to the tconst column in the basics table. This allows for linking alternative titles to their corresponding main titles.

For the basics table:
Primary key: tconst : The primary key is tconst. This is the unique identifier for each title and is necessary for data integrity and avoiding duplicate entries.

Foreign key : No foreign keys

For the crew table:
Primary key: tconst : The primary key is tconst. This is the unique identifier for each title and is necessary for data integrity and avoiding duplicate entries.

Foreign key: The foreign key is tconst, which refers to the tconst column in the basics table. This allows for linking crew information to the correct title.

For the episode table:
Primary key: tconst: The primary key is tconst. This is the unique identifier for each episode and is necessary for data integrity and avoiding duplicate entries.

Foreign key: The foreign key is parent_const, which refers to the tconst column in the basics table. This allows

for linking episodes to their corresponding TV series.

For ratings table:

Primary key: The primary key is tconst. This is the unique identifier for each title and is necessary for data integrity and avoiding duplicate entries.

Foreign key: The foreign key is tconst, which refers to the tconst column in the basics table. This allows for linking rating information to the correct title.

A. INSERTION

1)
```
INSERT INTO public.basics (tconst,
title_type, primary_title,
original_title, is adult, start year,
end_year, runtime_minutes, genres)
VALUES ('tt9999998', 'movie',
'New Movie','New Movie', false,
2023, NULL, 120,'Action, Drama')
```

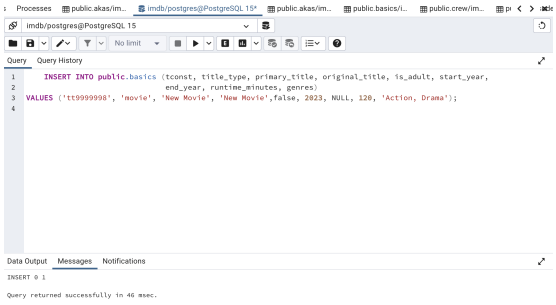


Fig. 2. Insert 1

2)
```
INSERT INTO crew (tconst, directors,
writers)
VALUES ('tt9999998', 'nm9876543',
'nm1234567, nm2345678');
```

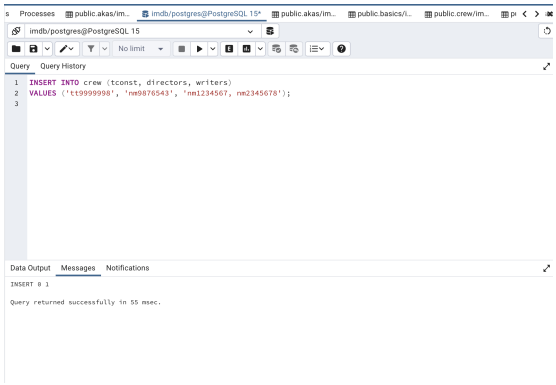


Fig. 3. Insert 2

B. UPDATION

1)
```
UPDATE akas
SET title = 'The Enchanted Inn',
region = 'US', types = 'working',
is_original_title true
WHERE title_id = 'tt0000018';
```

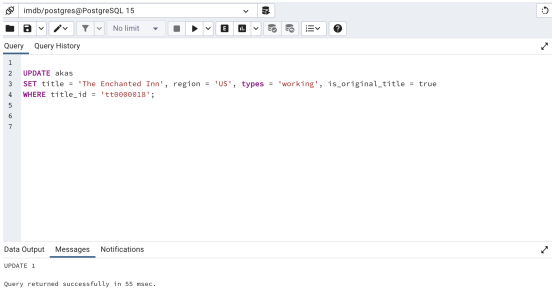


Fig. 4. Update 1

2)
```
UPDATE basics SET title_type = 'short',
primary_title = 'The Laughing Song',
original_title = 'Le rire',
is_adult = false, start_year = 1896,
end_year = NULL, runtime_minutes = 1,
genres = 'Comedy'
WHERE tconst = 'tt0000024';
```

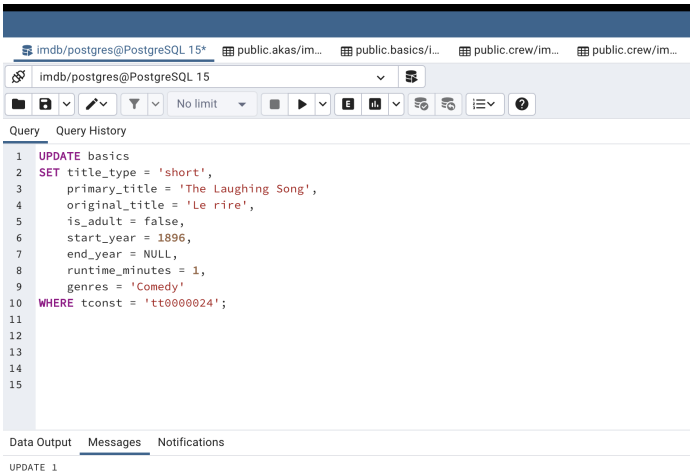


Fig. 5. Update 2

C. DELETION

1)
```
DELETE FROM akas WHERE
title_id = 'tt9999998' ;
```
2)
```
DELETE FROM ratings
WHERE tconst =  'tt9999998';
```

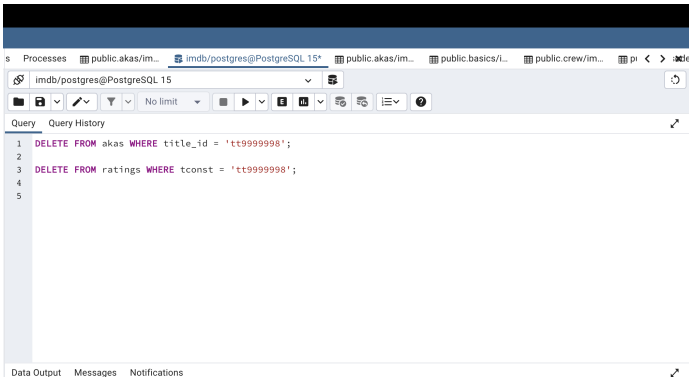


Fig. 6. Delete

D. SELECT QUERIES

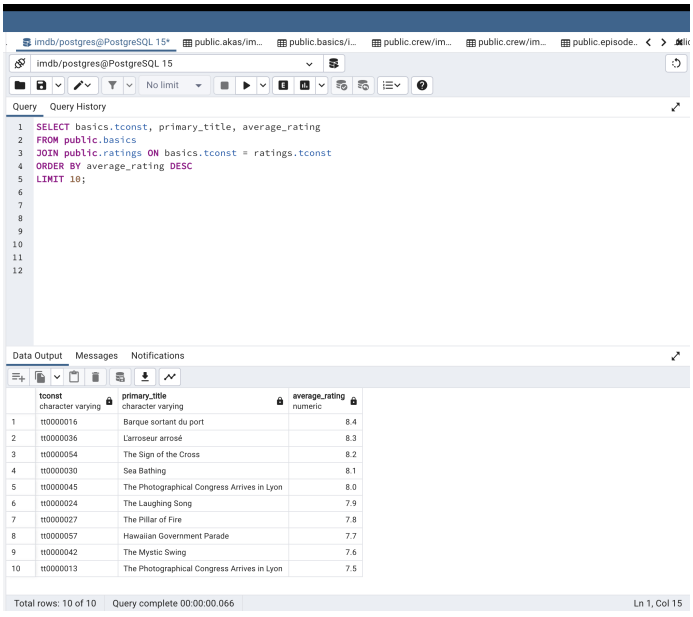
Q1: Find the top 10 highest-rated movies.

Q2: Sort genres by the highest number of movies

Q3: Number of movies released in a year along with ratings

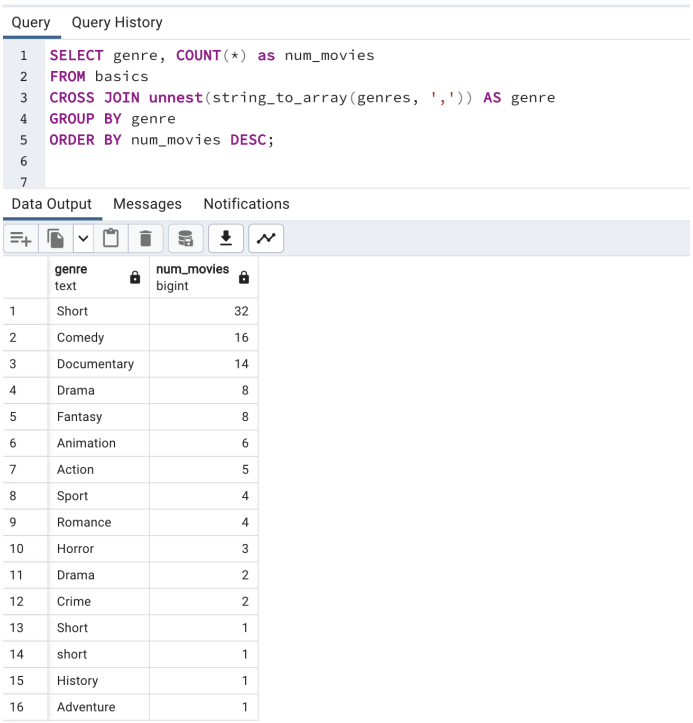
Q4: Episode and parent title information for a specific TV series, and sort by year and title

Q5: Give all comedy movies



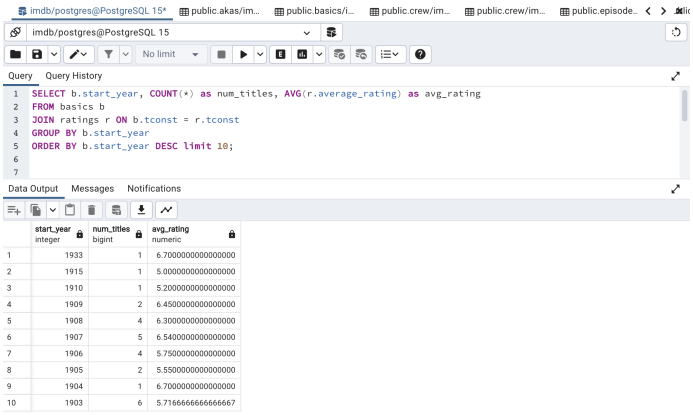
	tconst character varying	primary_title character varying	average_rating numeric
1	tt0000016	Barque sortant du port	8.4
2	tt0000036	Larroseur arrosé	8.3
3	tt0000054	The Sign of the Cross	8.2
4	tt0000030	Sea Bathing	8.1
5	tt0000045	The Photographical Congress Arrives in Lyon	8.0
6	tt0000024	The Laughing Song	7.9
7	tt0000027	The Pillar of Fire	7.8
8	tt0000057	Hawaiian Government Parade	7.7
9	tt0000042	The Mystic Swing	7.6
10	tt0000013	The Photographical Congress Arrives in Lyon	7.5

Fig. 7. Q1 : Top 10 highest rated movies



	genre text	num_movies bigint
1	Short	32
2	Comedy	16
3	Documentary	14
4	Drama	8
5	Fantasy	8
6	Animation	6
7	Action	5
8	Sport	4
9	Romance	4
10	Horror	3
11	Drama	2
12	Crime	2
13	Short	1
14	short	1
15	History	1
16	Adventure	1

Fig. 8. Q2 : Genres sorted by highest number of movies



	start_year integer	num_titles bigint	avg_rating numeric
1	1933	1	6.7000000000000000
2	1915	1	5.0000000000000000
3	1910	1	5.2000000000000000
4	1909	2	6.4500000000000000
5	1908	4	6.3000000000000000
6	1907	5	6.5400000000000000
7	1906	4	5.7500000000000000
8	1905	2	5.5500000000000000
9	1904	1	6.7000000000000000
10	1903	6	5.7166666666666667

Fig. 9. Q3 :Number of movies released in a year along with ratings

Fig 9. : This query joins the basics and ratings tables on their tconst columns and groups the results by the start_year column of the basics table. It then calculates the count of titles and the average rating for each year using the COUNT() and AVG() aggregate functions, respectively. Finally, it orders the results by the start_year column.

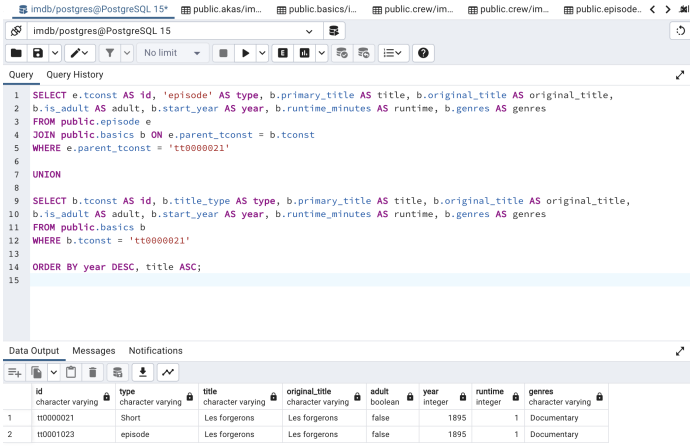


Fig. 10. Q4 :Episode and parent title information for a specific TV series, and sort by year and title along with ratings

Fig 10. : This query retrieves information about all episodes and the parent title of the TV series with ID 'tt0000021' from the episodes and basics tables. It selects the relevant columns and renames them according to the specifications given earlier. It uses UNION to combine the results of two queries and sorts the result by year (descending) and title (ascending).

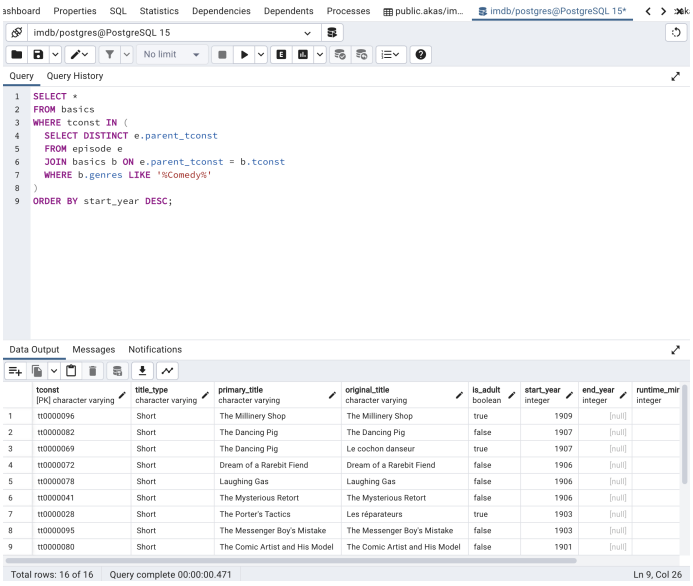


Fig. 11. Q5 :All comedy movies.

Fig 11. : This query will return all the TV series (parent titles) that have at least one episode and belong to the Comedy genre. The subquery first selects all the distinct parent_tconst values from the episode table that match these criteria, and then the outer query returns all the corresponding rows from the basics table, sorted by start_year in descending order

VII. Query Optimisation

In order to improve the performance of specific queries that may have taken longer to execute, indexing can be applied to them. For example, the execution time of a specific query can be reduced from 369ms to 142ms through indexing. For some queries, we followed different approaches to optimize them further. Refer to Fig. 12 and Fig. 13.

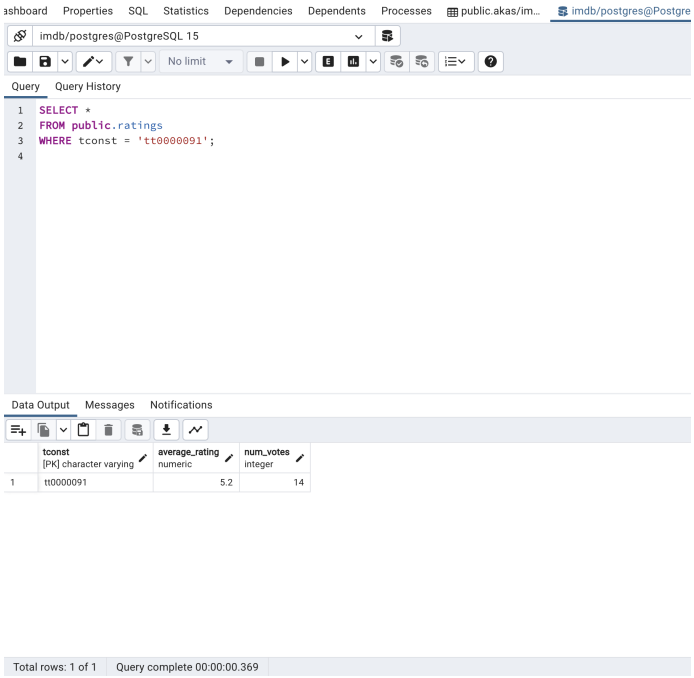


Fig. 12. Before Indexing

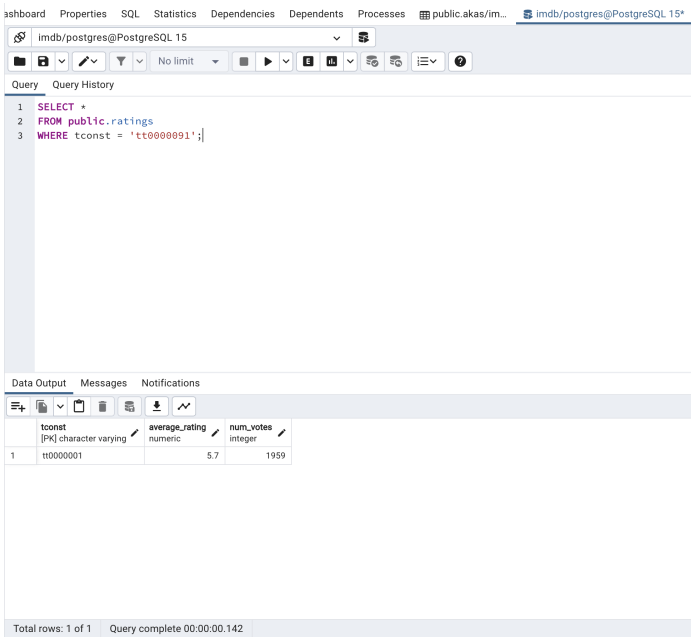


Fig. 13. After Indexing

In this query, We replaced the IN clause with an EXISTS clause, which can be faster for large subqueries. We also removed the join with the basics table in the subquery since we only need to check if the parent_tconst exists in the basics table. Refer to Fig. 14 and Fig. 15.

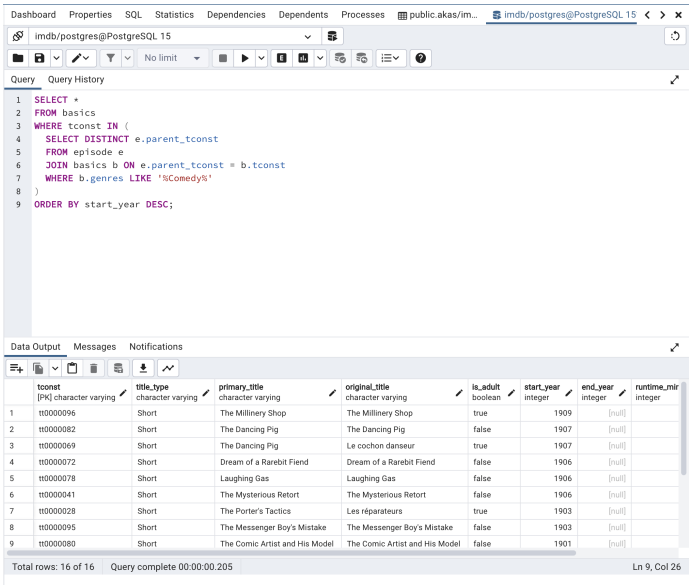


Fig. 14. Before Optimising

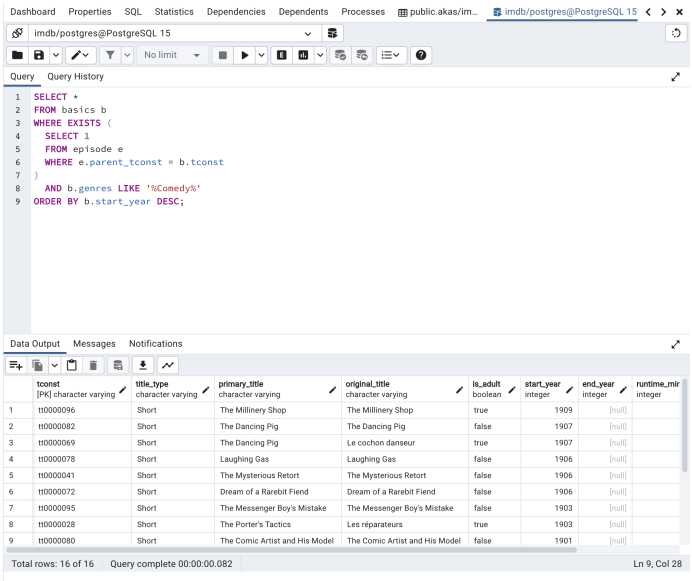


Fig. 15. After Optimising

In this query, We replaced the UNION operator with UNION ALL since we don't need to eliminate duplicates in this case. We also removed the unnecessary table aliases and used a single JOIN instead of two separate queries. Finally, I suggest creating an index on the parent_tconst column of the episode table to speed up the join operation. Refer to Fig. 16 and Fig. 17.

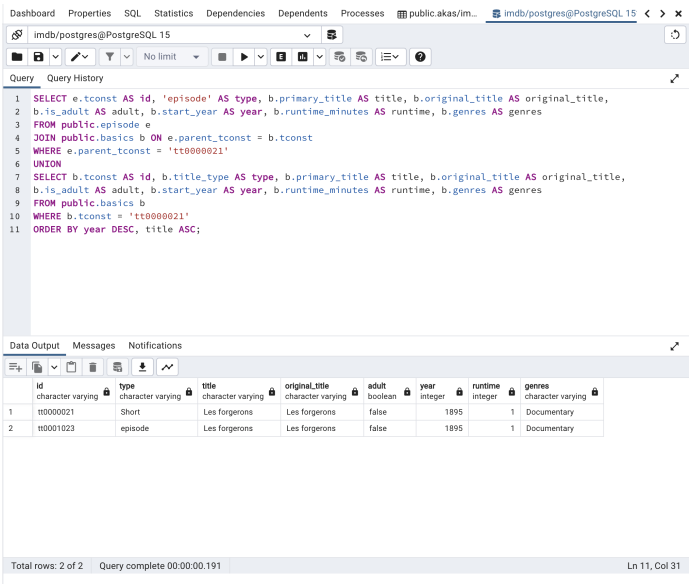


Fig. 16. Before Optimising

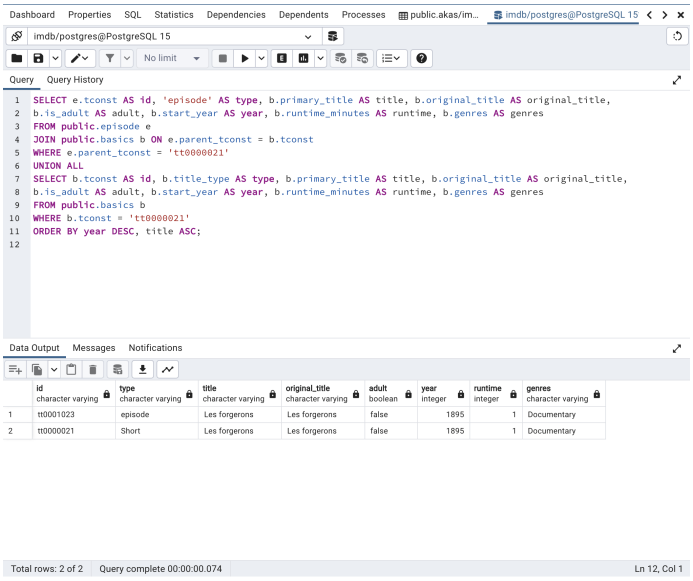


Fig. 17. After Optimising

VIII. BONUS TASK

The user interface (UI) includes a home page that displays the top 10 movies at the current moment by default. Additionally, there is a dropdown menu that allows users to filter results by categories such as top 10 movies by different genres like comedy, short, etc. Below is the screenshot that displays our home page

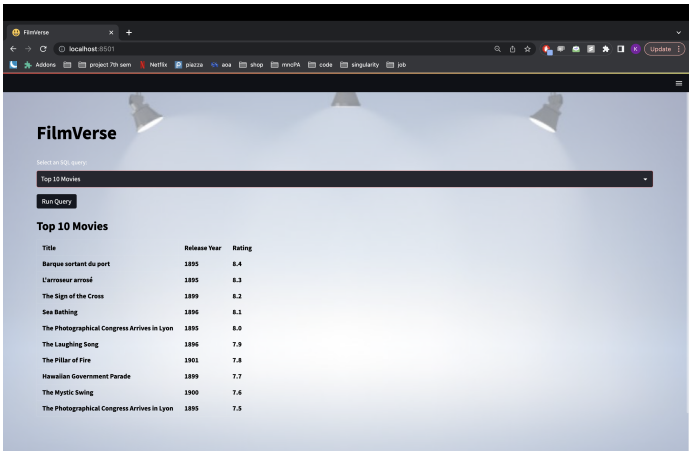


Fig. 18. UI Home Page

IX. TEAM CONTRIBUTIONS

Kajol

- Implemented Indexing
- Implemented Selection Queries
- Implemented frontend and backend for website
- Created Demo video

Mahimitra Chirala

- Problem Statement Selection
- Normalisation check and dependencies in BCNF
- Writing Simple Queries
- Created Demo video/Report review/ Testing

Sagar Gopalasetti

- Cleaning and modification of the dataset
- Creating Database
- ER diagram Implementation
- Report

REFERENCES

[1] The official IMDb datasets website: https://www.imdb.com/interfaces/
[2] IMDbPy - a Python package useful for retrieving and managing IMDb data: https://imdbpy.github.io/
[3] IMDb database schema: https://www.imdb.com/interfaces/IMDb%20Schema
[4] SQL queries for IMDb datasets: https://github.com/databasss/IMDb-SQL-Queries
[5] IMDb SQL database download: https://www.kaggle.com/stefanoleone992/imdb-extensive-dataset