IMDB Data Insights

Sai Yeshwanth Rachakonda[*], Ruhaizi Mopuri[†], Revanth Balineni[‡]
[*]*Department of Computer science*, [†]*Department of Computer science*, [‡]*Department of Computer science*
University at Buffalo, Buffalo, USA
{saiyeshw, ruhaizim, rbalineni}@buffalo.edu

***Abstract*—Hollywood has been releasing more and more Titles each year since its inception with an ever-increasing number of titles, and it becomes a hassle to keep all the attributes associated with the title in a structured form and filter titles based on specific requirements. Our project aims to provide these filtering capabilities to niche audiences who want titles aligned with their specific tastes.**

***Index Terms*—Normalization, Database, Movies, Relations, BCNF, Titles**

I. Problem Statement

In today's world of enormous amounts of data, it has become a challenge to store, organize and draw insights. Entertainment is an industry with such huge amounts of data related to movies and other online content. The Internet Movies Database (IMDb) provides information related to movies, podcasts, television series, video games, online streaming content, cast crew information, reviews, ratings etc. This is the most popular and reliable dataset related to entertainment. To process this data and provide relevant information to the relevant target groups is a challenging task. To address this issue, a sql database free of any kind of inconsistencies, redundancies is necessary.

A. Why do you need a database instead of an Excel file?

- A database scales the data more efficiently as the data grows while handling the data becomes slow and cumbersome with excel files.
- Simultaneous access to the data can be achieved through concurrency control mechanism.For any collaboration with proper version control, database gets the upper hand when compared to excel. This is because a shared excel file being accessed by multiple users can cause version control issues which can be avoided by using database.
- Data integrity is maintained in the database through triggers, constraints, and foreign key relationships. These consistencies can easily take place in the excel files.
- Authentication, encryption, access control are included in the database which provides security to protect the sensitive information.
- Excel provides only basic function to perform operations and extract information from the data. Database supports SQL which allow complex querying and analysis of data.
- Normalization of data is possible in databases, where the data is stored in an organized manner to maintain consistency and avoid anomalies. This task is challenging in excel files.
- Reliable backup and recovery solutions are offered by databases compared to excel files.
- Databases can be easily connected to other applications. The connector solutions offered by database are smoother and more effective making it a better choice instead of excels. They can integrate seamlessly and can help for any automation process setup for the application associated with the data from the database.
- Auditing and change management can be regulated and can follow standards when the data is stored in database and not in the excel file. Databases usually offer built in logging system which logs any transactions that happen through the database. These logs can help in the regular audits and in maintaining the change management database. This service is not provided in excel which can make it quite challenging to use for a project. This facilitates the organisations to stay compliant.

B. Discuss the background of the problem leading to your objectives. Why is it a significant problem?

- File Storage systems face scalability issues, redundancy issues, inconsistencies, and security problems. For huge amounts of data, the read and write operations become slow and makes the system inefficient. In today's world, there's a need of a centralized storage for data related to movies, TV shows, other entertainment related content to perform different kinds of operations and retrieve useful information for purposes such as movie recommendation, reviewing, feedback analysis, cast and crew information etc. the data provides comprehensive film information which is a valuable resource for people of entertainment industry and usual movie enthusiasts.
- This acts as a promotional platform for many movie makers. It holds the historical records of the film and tv industry providing release dates, cast, summaries and other information.
- Since there are many use cases and applications of this dataset, any kind of inconsistency in this data could lead to huge loss for the users or application who rely on this dataset. The data needs to provide useful insights through effective querying which is offered by sql.

C. Explain the potential of your project to contribute to your problem domain. Discuss why this contribution is crucial.

Providing the data with least inconsistencies and organized for effective querying in the entertainment industry is the main goal of the project. To achieve this, a centralized database with structured form of data following BCNF is provided to ensure reliable data access. The project can make a substantial contribution to the entertainment industry in following ways. Not just the entertainment industry, it can even help others associated with the entertainment industry like the movie schools, media outlets. Other industries like sports and lifestyle can also benefit from this information to learn about the audience viewership.

- **Personalized Recommendations:** based on the user's interests, preferences and history, utilizing the vast amount of data related to television shows, movies and celebrities and other entertainment content.Online platforms can feed their algorithms to display and promote the television shows, movies and other content and can prioritise them accordingly.
- **Informed-decision-making:** By providing the useful information about the movies and TV series will actual give the user an idea to about the movie or series and helps them in making a decision to watch it or not.Most of the users use this data to decide where to invest their time in.
- **Data analysis and insights:** The data we are providing and the analysis we are doing will be useful for the professionals and researchers to analyze the audience preferences like what audience what and work according to it.It can help them understand the genre, language, and the platform most of the users prioritising. With the introduction of Over-the-top (OTT), there's a big shift in the viewership from televisions, cinema halls. This shift can be analysed and necessary decisions about the distribution can be taken preventing any loss due to the platform.

II. Target User

A. *Who will use your database?*

- This database can be used by a range of audiences including movie buffs, movie critics and various other users such as actors, directors, distributors, production houses, storywriters.
- This database can be used to study the trend in audience inclination and change in their preferences over time. It can help critics and movie producers to understand the box office collection and comment on them.
- A regular movie enthusiast can use this database to know the genre, rating, and other factors to decide on the film to watch. Directors can use this database to make critical decisions regarding their films about the actors they want to choose based on the genre they are greatly associated with.
- This can also help the distributors to decide on the platform they want to release their film on. A storywriter can decide on the type of story to write which can resonate with the audience pulse.
- Streaming platforms like Amazon prime, Netflix and others can use this database to train their recommendation algorithms which can increase the user screen time and in turn increase the viewership on their platform.
- Movie schools and students can utilize the data provided by this database to understand and filmmaking work by different filmmakers and to study the genres. Every filmmaker have a unique sense of identity which can be observed in the data from the database.
- Media Outlet companies can use this database to analyse, draw statistics and come to conclusions with probabilistic reasoning to generate news for their customers. Most of the publishers and news outlets utilise these statistics to create web articles and even publish the news in newspapers which have a dedicated section for Entertainment.
- Podcasts are a raising trend amongst the audience today and with the amount of podcasters coming into limelight, it is only expected that they can make a good use of this database for their profession. Podcasts viewership is mainly dependent on the topic of interest chosen and by choosing the right and interesting topic would not only increase the followers but also the subscription. Hence podcasters can analyse this data to find audience's interest and can design their podcasts based on them.

B. *Who will administer the database? real-life scenario*

- Software Engineers with the knowledge and skill of database administration can administer the database based on the interests of the companies who possess this.
- They will have the skill to update, delete, alter, and maintain the database. One of a real-life scenario can be a database administrator working for a production house company. He/She can analyze the data from this database and can provide this analysis to the higher management to determine which director's movie they want to invest their resources in and plan their business accordingly.
- And eventually the database administrator can keep updating the database to have the latest trend and up-to-date statistics of the movie market. This will help them to learn audience preferences. A media outlets company can have their database administrator analyze the database to draw statistics and create news for their customers.
- A media outlets company can have their database administrator analyze the database to draw statistics and create news for their customers. These statistics can be reported to the higher officials who can decide on the news that have to be published. By doing this, they can pique the interests of their customers to follow their channel be it online or offline.

Data Source: We have selected the IMDB dataset which was given in the project description file. Here is the link for the chosen website https://developer.imdb.com/non-commercial-datasets/

III. Proving that the relations we have are in BCNF

We say a relational schema R having functional dependencies F is said to be in BCNF form if and only if for all the functional dependencies in F+ which are in the form of $a \rightarrow b$ follows any of the following condition:

- $a \rightarrow b$ is trivial.
- a is the super key for the relation R.
  In our database the functional dependencies in all the tables (relations) and are given as below and we have also found out the candidate key for all the relations.And prove that the relations are in BCNF form

title.timeline

- $tconst \rightarrow startyear$
- $tconst \rightarrow endyear$
- tconst is the primary key which implies that it is the super key. Since all the values on the right hand side in the functional dependencies are determined by the super key, the table title.timeline is in BCNF.

title.akas

- One of the candidate keys of the title.akas table is "ordering" and the functional dependencies having this attribute are.
- $titleid, ordering \rightarrow title$
- $titleid, ordering \rightarrow region$
- $titleid, ordering \rightarrow language$
- $titleid, ordering \rightarrow type$
- $titleid, ordering \rightarrow attribute$
- $titleid, ordering \rightarrow isoriginaltitle$
- in the title.akas table, the primary key is a composite key constituting titleid and ordering. Both the values combined uniquely identify the rows of the table. (Titleid, ordering) is also super key which indicates that the remaining attributes in the table are determined by the super key. Thus the table is in BCNF.

title.basics

- $tconst \rightarrow titleType$
- $tconst \rightarrow primaryTitle$
- $tconst \rightarrow originalTitle$
- $tconst \rightarrow isAdult$
- $tconst \rightarrow runtimeMinutes$
- $tconst \rightarrow genre$
- tconst is the primary key which implies that it is the super key. Since all the values on the right hand side in the functional dependencies are determined by the super key, the table title.basics is in BCNF.

title.director

- $tconst, director \rightarrow director$
- the title.director has a trivial dependency i.e., director is determined by the primary key (tconst, director) and director is a subset of the primary key. Thus the table is in BCNF.

title.writer

- $tconst, writer \rightarrow writer$
- the title.writer has a trivial dependency i.e., writer is determined by the primary key (tconst, writer) and director is a subset of the primary key. Thus the table is in BCNF.

title.episode

- $tconst \rightarrow parentTconst$
- $tconst \rightarrow seasonNumber$
- $tconst \rightarrow episodeNumber$
- tconst is the primary key which implies that it is the super key. Since all the values on the right hand side in the functional dependencies are determined by the super key, the table title.episode is in BCNF.

title.principals

- $tconst, ordering \rightarrow nconst$
- $tconst, ordering \rightarrow category$
- $tconst, ordering \rightarrow job$
- $tconst, ordering \rightarrow characters$
- in the title.principals table, the primary key is a composite key constituting tconst and ordering. Both the values combined uniquely identify the rows of the table. (tconst, ordering) is also super key which indicates that the remaining attributes in the table are determined by the super key. Thus the table is in BCNF.

title.ratings

- $tconst \rightarrow averageRating$
- $tconst \rightarrow numVotes$
- tconst is the primary key which implies that it is the super key. Since all the values on the right hand side in the functional dependencies are determined by the super key, the table title.ratings is in BCNF.

name.basics

- $nconst \rightarrow primaryName$
- $nconst \rightarrow birthYear$
- $nconst \rightarrow deathYear$
- nconst is the primary key which implies that it is the super key. Since all the values on the right hand side in the functional dependencies are determined by the super key, the table name.basics is in BCNF.

worker.basics relation

- $nconst \rightarrow primaryProfession$
- $nconst \rightarrow knownForTitles$
- nconst is the primary key which implies that it is the super key. Since all the values on the right hand side in the functional dependencies are determined by the super key, the table worker.basics is in BCNF.

IV. Tables and their primary keys, attributes and foreign keys

- **Action taken on deletion:** we are cascading on delete that is we are deleting all the references in the child table as well when the instances are deleted from the parent table. On deleting references in the child table there wont be any changes in the parent table
- **All the values default values are set "NULL" for all the relations**
- There are no primary key constarints set. So, when we try to delete a row which refers to other values in different relation the statement will not execute and returns us an error i.e., the cascade is not being set

A. The attributes and the description of attributes of the title.basics table are as follows:

- tconst - It is a alphanumeric value that uniquely identifies the title. The domain of this is Varchar
- titleType - Type of the title like movie or tvseries or tvepisode. The domain of this is Varchar
- primaryTitle – The more popular title or the titles used at the point of release. The domain of this is Varchar
- originalTitle - Original title. The domain of this is Varchar
- isAdult - The domain of this attrubute is boolean i.e., 0 for non adult and 1 for adult movie
- runtimeMinutes – primary runtime of the title, in minutes. The domain of this attribute is integer
- genre – This states the genre of the film or the TV series.The domain of this attribute is Varchar
- The primary key is tconst
- There is no foreign key for title.basics

B. The attributes and the description of attributes of the title.timeline table are as follows:

- startYear (YYYY) – represents the release year of a title. In the case of TV Series, it is the series start year. The domain of this attribute integer
- endYear (YYYY) – TV Series end year.
  N for all other title types. The domain of this attribute is integer
- primary key is tconst and it refers to tittle.basics

C. The attributes and the description of attributes of the title.writer table are as follows:

- tconst - It is a alphanumeric value that uniquely identifies the title. The domain of this is Varchar
- writer – Writer of the given title. The domain of this is Varchar
- primary key is (tconst,writer) and tconst refers to tconst from table.basics table

D. The attributes and the description of attributes of the title.episode table are as follows:

- tconst - It is a alphanumeric value that uniquely identifies the title. The domain of this is Varchar
- parentTconst - alpha numeric value which identifies the Parent TV series The domain of this is Varchar.
- seasonNumber – Season number the episode belongs to.The domain of this is integer
- episodeNumber – episode number of the tconst in the TV series.The domain of this is integer
- The Primary key for this relation is tconst, (tconst, parenttconst) refer to table.basics

E. The attributes and the description of attributes of the title.principals table are as follows:

- tconst - It is a alphanumeric value that uniquely identifies the title. The domain of this is Varchar
- ordering - a number to uniquely identify rows for a given titleId. The domain of this is integer
- nconst – alphanumeric unique identifier of the name/person. The domain of this attribute is Varchar.
- category – the category of job that person was in. The domain of this is varchar.
- job – the specific job title if applicable, else '
  N'. The domain of this is varchar.
- characters – the name of the character played if applicable, else '
  N'. The domain of this is varchar.
- the primary key for this is (tconst, ordering), the foreign key is nconst from name.basics and tconst refers to tconst from table.basics tabl

F. The attributes and the description of attributes of the title.ratings table are as follows:

- tconst - It is a alphanumeric value that uniquely identifies the title. The domain of this is Varchar
- averageRating - weighted average of all the individual user ratings. The domain of this is integer
- numVotes – number of votes the title has received. The domain of this attribute is Integer.
- The primary key is tconst and it refers to tconst from table.basics table

G. The attributes and the description of attributes of the name.basics table are as follows:

- nconst - alphanumeric unique identifier of the name/person. The domain of this is Varchar
- primaryName - name by which the person is most often credited. The domain of this is Varchar
- birthYear – Birth year of the person. The domain of this attribute is Integer.
- deathYear – Death year of the person if not applicable
  N. The domain of this attribute is Integer.
- The primary key is nconst and there is no foreign key

H. The attributes and the description of attributes of the worker.basics table are as follows:

- nconst - alphanumeric unique identifier of the name/person. The domain of this is Varchar
- primaryProfession - The primary profession of the worker. The domain of this is Varchar

- KnownForTitle – Title the person known for. The domain of this attribute is Varchar.
- The primary key is nconst and it refers to nconst from name.basics table

I. The attributes and the description of attributes of the title.akas table are as follows:

- titleId - a tconst, an alphanumeric unique identifier of the title. The domain of this is Varchar
- ordering - a number to uniquely identify rows for a given titleId. The domain of this is Integer
- title – the localized title. The domain of this attribute is Varchar.
- region - the region for this version of the title.The domain of this attribute is Varchar.
- language - the language of the title.The domain of this attribute is Varchar.
- type - attribute for the alternative title. The domain of this attribute is Varchar.
- attribute - term required to describe alternative title. The domain of this attribute is Varchar.
- isOriginalTitle - 0 for original title and 1 for original title. The domain of this attribute if boolean
- The primary key for this relation is (titleid, ordering) and titleId is a foreign key for this relation and it refers to tconst from title.basics

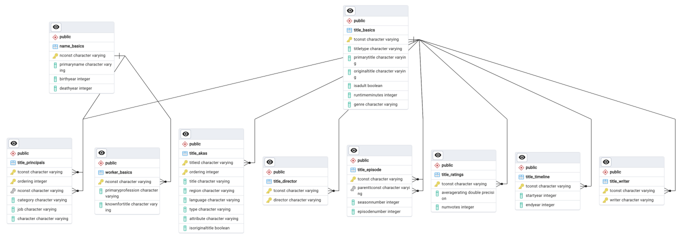
J. The attributes and the description of attributes of the title.director table are as follows:

- tonst - a tconst, an alphanumeric unique identifier of the title. The domain of this is Varchar
- director - Attribute specifies the name of the director of the title. The domain of this attribute of varchar
- primary key is (tconst,director), and foreign key tconst references to title.basics

V. Relations between different tables

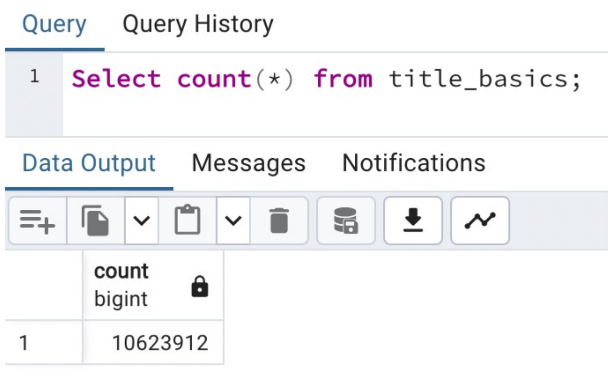
- **Title.basics:** We have chosen tconst as the Primary Key this is because tconst is a unique value, the rest of the columns are not unique
- **Title.akas:** A single title might have multiple akas. For instance for Avengers: Age of Ultron it is Avengers 2, Avengers: Rise of Ultron, this creates a issue where we cant use only tconst as aka. So we generated ordering column which increments automatically for each time the tconst is referenced the combination of (tconst, ordering) is unique hence can be used as primary key.
- **Title.principals:** The principal people who have worked on the Title are given entries in this table. A movie can have multiple people working on it. So, we've followed a similar approach to akas and have chosen a combination of (tconst, ordering) for the table title.principals
- **Name.basics:** The personal details of the actors, writers, and production staff are in this table. The nconst column in this tab;e is unique and is the primary key.
- **Worker.basics:** The professions and other prominent project the actors/production staff has worked on is stored in the table. Since this describes the entities of the name.basics the primary key of the former can be referred to in this table as the primary key.
- **Title.director:** The directors associated with a movie are stored in the table. A single tconst can have multiple directors. So a combination of (tconst, director) will be unique and hence can be used as the primary key.
- **Title.writer:** The writers associated with a movie are stored in the table. A single tconst can have multiple writers. So a combination of (tconst, director) will be unique and hence can be used as the primary key.
- **Title.episode:** A grouping of episodes by parent show's tconst. The parenttconst and tconst refer to the title.basics table with foreign key constraints as they themselves have rating episode wise and season-wise. Ex: Perfect Game episode in Attack on Titan has rating of 9.7/10, Whole Attack on Titan series has rating of 9.1/10.
- **Title.ratings:** The rating and no of votes for each tconst. Hence tconst is referenced to from title.basics as primary key.
- **Title.timeline:** Has timeline associated with each tconst hence references to the same from title.basics as primary key.

VI. ER Diagram



-

VII. Questions and Solutions



Query Query History

```
Select count(*) from title_basics;
```

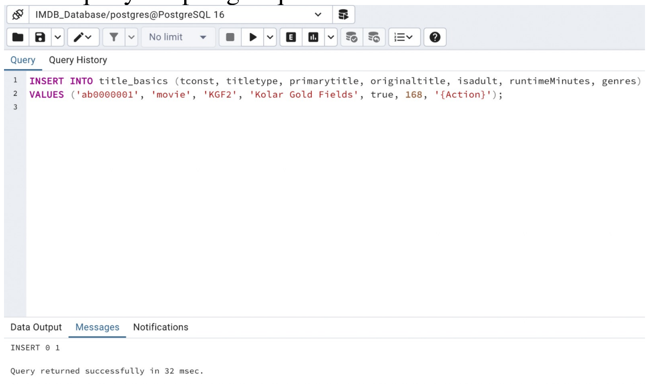
Data Output Messages Notifications

	count bigint
1	10623912

-
- Our dataset consists of large amount of rows which is around 10Million so we have to solve the problem of long retrieval times and make the query more efficient.
- **Question: How to efficiently deal with 10 Million rows**

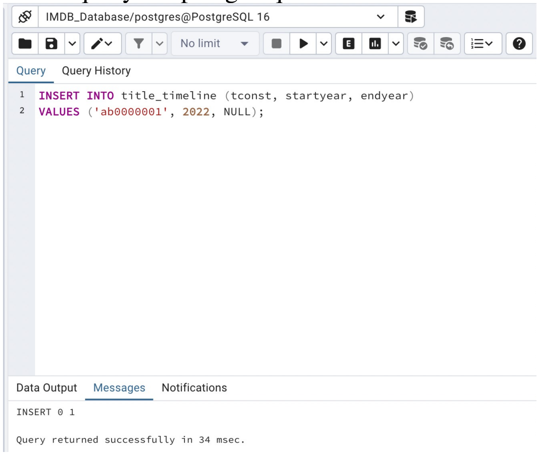
- So to solve the above mentioned problems in our database we have implemented the feature of indexing which will help us in data retrieval. It is very similar to the index in a book where we can easily find what we are looking for when we first look at the index.
- In our database we have implemented the feature of indexing which will help us in data retrieval. It is very similar to the index in a book where we can easily find what we are looking for when we first look at the index.
- **Fast data retrieval** Without indexing, the database has to go through every row to find the desired information. But with indexes, it is very easy to locate the desired row for the database by going through the index structure. This will reduce the data retrieval time.
- **Optimized query performance** Indexes act like roadmaps for the database, helping it find information quickly. When you ask a question (query) of the database, it checks these roadmaps to see the most efficient way to get the answer. This allows the database to find the information you need much faster.
- **Question: How to deal with foreign key violations when loading 10 tables**
- To solve the foreign key violation problems that is when we are trying to load the data in the child table which is referencing tuples which are not existent in the parent table. This is due to referential integrity, which means every value in the foreign key column of the child table must have a primary key value in the referenced parent table.
- we have solved this problem by loading the data into a staged table that is first we have loaded the data into a table with no foreign key constraints the after doing the necessary transformations and changes we then loaded the data into the real tables.

VIII. Basic Queries

- **Query 1:** INSERT INTO title_basics (tconst, titletype, primarytitle, originaltitle, isadult, runtimeMinutes, genres) VALUES ('ab0000001', 'movie', 'KGF2', 'Kolar Gold Fields', true, 168, 'Action');
- here we are trying to insert a record into title_basics table that is we are trying to insert details about a movie title. Below is the attached image showing the execution of the above query on postgresql
- 

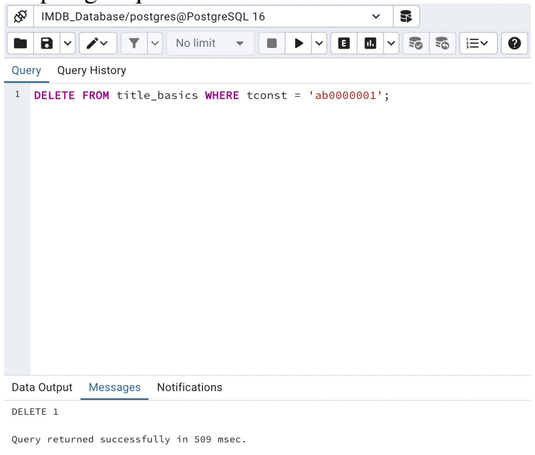
```
INSERT INTO title_basics (tconst, titletype, primarytitle, originaltitle, isadult, runtimeMinutes, genres)
VALUES ('ab0000001', 'movie', 'KGF2', 'Kolar Gold Fields', true, 168, '{Action}');
```

```
INSERT 0 1

Query returned successfully in 32 msec.
```

- **Query 2:** INSERT INTO title_timeline (tconst, startyear, endyear) VALUES ('ab0000001', 2022, NULL);
- here we are trying to insert a record into title_timeline table that is we are trying to insert details about a movie like which year it has started and which year it has ended. Below is the attached image showing the execution of the above query on postgresql
- 

```
INSERT INTO title_timeline (tconst, startyear, endyear)
VALUES ('ab0000001', 2022, NULL);
```

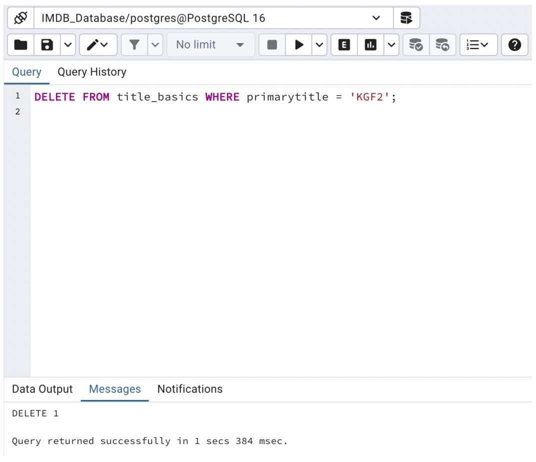
```
INSERT 0 1

Query returned successfully in 34 msec.
```

- **Query 3:** DELETE FROM title_basics WHERE tconst = 'ab0000001';
- here we are trying to delete a recocrd from the title_basics table where the tconst = 'ab0000001'. Below is the attached image showing the execution of the above query on postgresql
- 

```
DELETE FROM title_basics WHERE tconst = 'ab0000001';
```

```
DELETE 1

Query returned successfully in 509 msec.
```

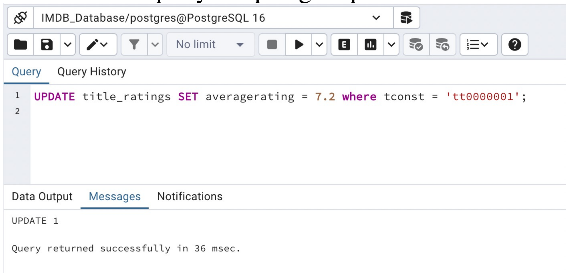
- **Query 4:** DELETE FROM title_basics WHERE primarytitle = 'KGF2';
- here we are trying to delete a recocrd from the title_basics table where the primarytitle = 'KGF2'. Below is the attached image showing the execution of the above query on postgresql

- 

```
DELETE FROM title_basics WHERE primarytitle = 'KGF2';
```

```
DELETE 1

Query returned successfully in 1 secs 384 msec.
```

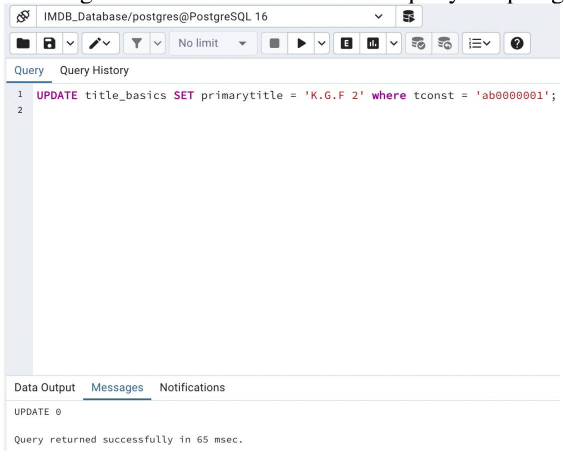
- **Query 5:** UPDATE title_ratings SET averagerating = 7.2 where tconst = ’tt0000001’;
- here we are trying to update a record in the update_title which has tconst as ’tt0000001’ and set the averagerating as 7.2. Below is the attached image showing the execution of the above query on postgresql

- 

```
UPDATE title_ratings SET averagerating = 7.2 where tconst = 'tt0000001';
```

```
UPDATE 1

Query returned successfully in 36 msec.
```

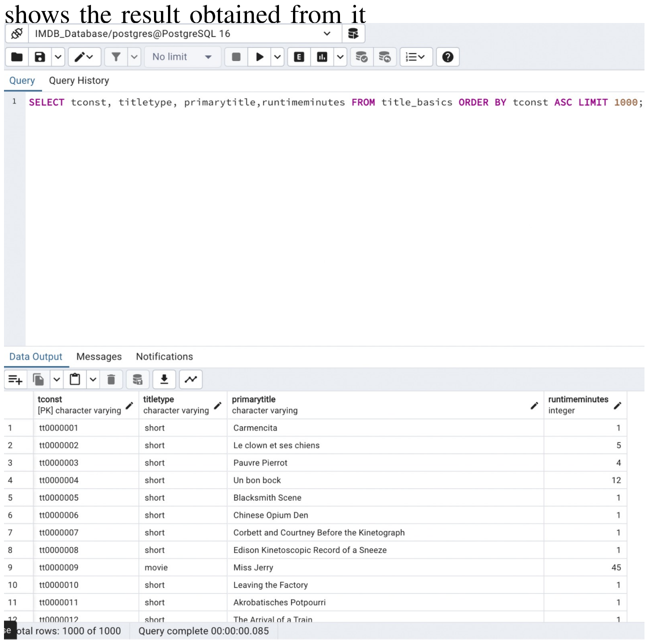
- **Query 6:** UPDATE title_basics SET primarytitle = ’K.G.F 2’ where tconst = ’ab0000001’;
- here we are trying to update a record in the update title_basics which has tconst as ’ab0000001’ and set the primarytitle = ’K.G.F 2’. Below is the attached image showing the execution of the above query on postgresql

- 

```
UPDATE title_basics SET primarytitle = 'K.G.F 2' where tconst = 'ab0000001';
```

```
UPDATE 0

Query returned successfully in 65 msec.
```

- **Query 7:** SELECT tconst, titletype, primarytitle,runtimeminutes FROM title_basics ORDER BY tconst ASC LIMIT 1000;
- Here we are trying to get data from the title_basics table and we are only getting the columns tconst, titletype,primarytitle,runtimeminutes. Here we are also ordering the data based on the tconst values and we are limiting the numbe of rows to only 1000 that is we will be getting 1000 rows of data. Below is the attached image showing the execution of the above query on postgresql and also shows the result obtained from it

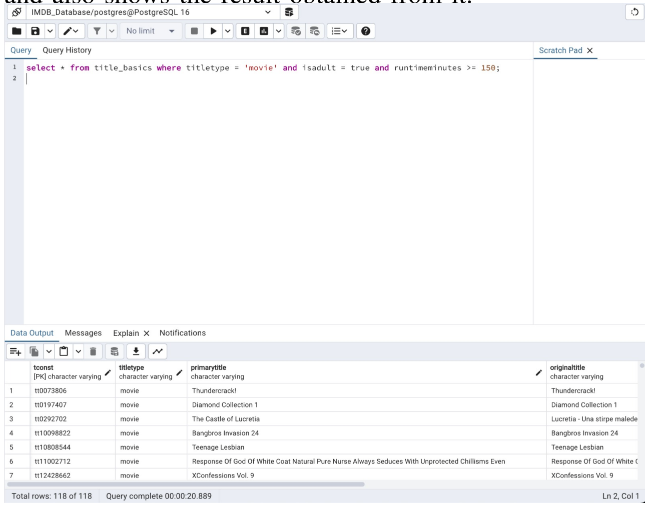
- 

```
SELECT tconst, titletype, primarytitle,runtimeminutes FROM title_basics ORDER BY tconst ASC LIMIT 1000;
```

	tconst [PK] character varying	titletype character varying	primarytitle character varying	runtimeminutes integer
1	tt0000001	short	Carmencita	1
2	tt0000002	short	Le clown et ses chiens	5
3	tt0000003	short	Pauvre Pierrot	4
4	tt0000004	short	Un bon bock	12
5	tt0000005	short	Blacksmith Scene	1
6	tt0000006	short	Chinese Opium Den	1
7	tt0000007	short	Corbett and Courtney Before the Kinetograph	1
8	tt0000008	short	Edison Kinetoscopic Record of a Sneeze	1
9	tt0000009	movie	Miss Jerry	45
10	tt0000010	short	Leaving the Factory	1
11	tt0000011	short	Akrobatisches Potpourri	1
12	tt0000012	short	The Arrival of a Train	1

Total rows: 1000 of 1000 Query complete 00:00:00.085

- **Query 8:** SELECT * from titlebasics where titletype = ’movie’ and isadult = true and runtimeminutes >150;
- Here we are trying to fetch adult movies whose run time is greater than 150 minutes. Below is the attached image showing the execution of the above query on postgresql and also shows the result obtained from it.

- 

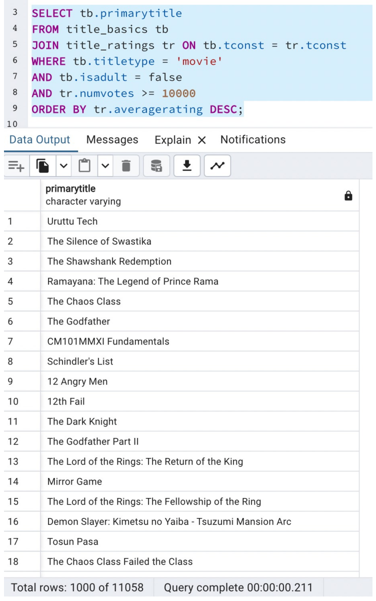
```
select * from title_basics where titletype = 'movie' and isadult = true and runtimeminutes >= 150;
```

	tconst [PK] character varying	titletype character varying	primarytitle character varying	originaltitle character varying
1	tt0073806	movie	Thundercrack!	Thundercrack!
2	tt0197407	movie	Diamond Collection 1	Diamond Collection 1
3	tt0292702	movie	The Castle of Lucretia	Lucretia - Una stirpe malede
4	tt10098822	movie	Bangbros Invasion 24	Bangbros Invasion 24
5	tt10808544	movie	Teenage Lesbian	Teenage Lesbian
6	tt11002712	movie	Response Of God Of White Coat Natural Pure Nurse Always Seduces With Unprotected Chillisms Even	Response Of God Of White (
7	tt12428662	movie	XConfessions Vol. 9	XConfessions Vol. 9

Total rows: 118 of 118 Query complete 00:00:20.889 Ln 2, Col 1

- **Query 9:** SELECT tb.primary_title FROM title_basics tb JOIN title_ratings tr ON tb.tconst = tr.tconst WHERE tb.title_type = ’movie’ AND tb.is_adult = false AND tr.numvotes >= 10000 ORDER BY tr.averagerating DESC;
- Here we are trying to fetch primary_title of non-adult movies in descending order of their average rating from

the title_basics table and title_ratings table, which have minimum 10000 number of votes. Below is the attached image showing the execution of the above query on postgresql and also shows the result obtained from it.



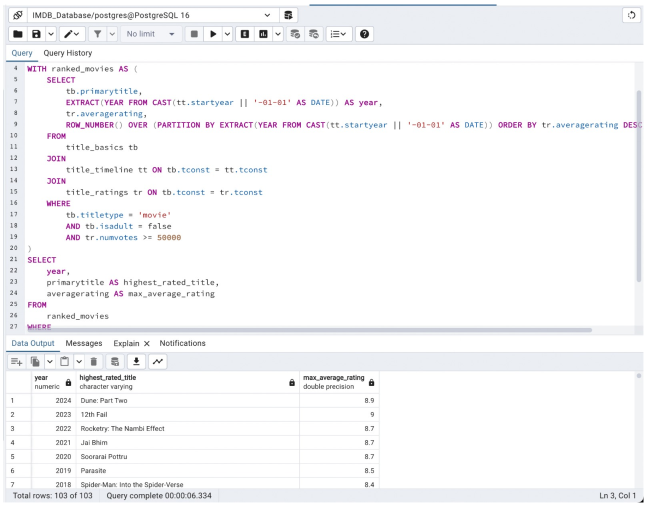
```sql
SELECT tb.primarytitle
FROM title_basics tb
JOIN title_ratings tr ON tb.tconst = tr.tconst
WHERE tb.titletype = 'movie'
AND tb.isadult = false
AND tr.numvotes >= 10000
ORDER BY tr.averagerating DESC;
```

Data Output | Messages | Explain | Notifications

	primarytitle character varying
1	Uruttu Tech
2	The Silence of Swastika
3	The Shawshank Redemption
4	Ramayana: The Legend of Prince Rama
5	The Chaos Class
6	The Godfather
7	CM101MMXI Fundamentals
8	Schindler's List
9	12 Angry Men
10	12th Fail
11	The Dark Knight
12	The Godfather Part II
13	The Lord of the Rings: The Return of the King
14	Mirror Game
15	The Lord of the Rings: The Fellowship of the Ring
16	Demon Slayer: Kimetsu no Yaiba - Tsuzumi Mansion Arc
17	Tosun Pasa
18	The Chaos Class Failed the Class

Total rows: 1000 of 11058 Query complete 00:00:00.211

- **Query 10:** WITH ranked_movies AS (SELECT tb.primary_title, EXTRACT(YEAR FROM CAST(tt.start_year ||'-01-01' AS DATE)) AS year,tr.averagerating,ROWNUMBER() OVER (PARTITION BY EXTRACT(YEAR FROM CAST(tt.startyear ||'-01-01' AS DATE)) ORDER BY tr.averagerating DESC) AS rn FROM title_basics tb JOIN title_timeline tt ON tb.tconst = tt.tconst JOIN title_ratings tr ON tb.tconst = tr.tconst WHERE tb.titletype = 'movie' AND tb.isadult = false AND tr.numvotes >= 50000)
SELECT year, primarytitle AS highest_rated_title, averagerating AS max_average_rating FROM ranked_movies WHERE rn = 1 ORDER BY year DESC;
- Here we are trying to fetch year, highest_rated_title and max_rating of non-adult movie with a minimum number of votes of 50000, in each year in descending order of the year they were released from title_basics, title_timeline and titile_ratings tables by joining the tables and adding the constraints.



```sql
WITH ranked_movies AS (
    SELECT
        tb.primarytitle,
        EXTRACT(YEAR FROM CAST(tt.startyear || '-01-01' AS DATE)) AS year,
        tr.averagerating,
        ROW_NUMBER() OVER (PARTITION BY EXTRACT(YEAR FROM CAST(tt.startyear || '-01-01' AS DATE)) ORDER BY tr.averagerating DESC
    FROM
        title_basics tb
    JOIN
        title_timeline tt ON tb.tconst = tt.tconst
    JOIN
        title_ratings tr ON tb.tconst = tr.tconst
    WHERE
        tb.titletype = 'movie'
        AND tb.isadult = false
        AND tr.numvotes >= 50000
)
SELECT
    year,
    primarytitle AS highest_rated_title,
    averagerating AS max_average_rating
FROM
    ranked_movies
WHERE
```

Data Output | Messages | Explain | Notifications

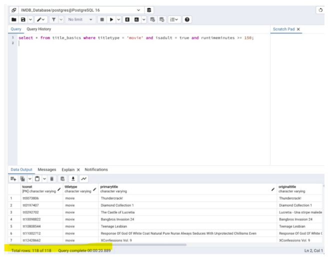
	year numeric	highest_rated_title character varying	max_average_rating double precision
1	2024	Dune: Part Two	8.9
2	2023	12th Fail	9
3	2022	Rocketry: The Nambi Effect	8.7
4	2021	Jai Bhim	8.7
5	2020	Soorarai Pottru	8.7
6	2019	Parasite	8.5
7	2018	Spider-Man: Into the Spider-Verse	8.4

Total rows: 103 of 103 Query complete 00:00:06.334 Ln 3, Col 1

IX. Problematic Queries and their Performance Improvement

Following are some problematic queries which are consuming excessive amount of time to execute. To address this issue, we implemented indexing on few columns of some relations. through indexing, the database can easily locate the rows based on indexed column values.

- **Problematic Query 1:**



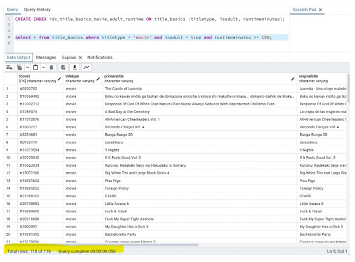
```sql
select * from title_basics where titletype = 'movie' and isadult = true and runtimeminutes >= 150;
```

Data Output | Messages | Explain | Notifications

	tconst [PK] character varying	titletype character varying	primarytitle character varying	originaltitle character varying
1	tt0073806	movie	Thundercrack!	Thundercrack!
2	tt0197407	movie	Diamond Collection 1	Diamond Collection 1
3	tt0292702	movie	The Castle of Lucretia	Lucretia - Una stirpe malede
4	tt10098822	movie	Bangbros Invasion 24	Bangbros Invasion 24
5	tt10808544	movie	Teenage Lesbian	Teenage Lesbian
6	tt11002712	movie	Response Of God Of White Coat Natural Pure Nurse Always Seduces With Unprotected Chillisms Even	Response Of God Of White (
7	tt12428662	movie	XConfessions Vol. 9	XConfessions Vol. 9

Total rows: 118 of 118 Query complete 00:00:20.889 Ln 2, Col 1

As can be observed from the screenshot, the query is taking 20 seconds to fetch the data, which indicates slow performance. To reduce the execution time, we implemented indexing on (titletype, isadult, runtimeminutes) columns in title_basics relation. This way, the data is not fetched in sequential manner and indexing are used instead.



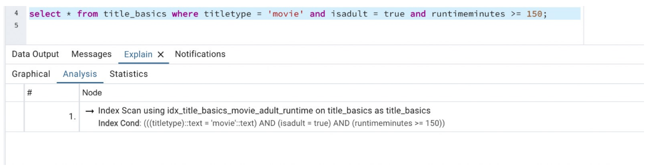
```sql
CREATE INDEX idx_title_basics_movie_adult_runtime ON title_basics (titletype, isadult, runtimeminutes);


select * from title_basics where titletype = 'movie' and isadult = true and runtimeminutes >= 150;
```

Data Output | Messages | Explain | Notifications

	tconst [PK] character varying	titletype character varying	primarytitle character varying	originaltitle character varying
1	tt0292702	movie	The Castle of Lucretia	Lucretia - Una stirpe malede
2	tt10369492	movie	Boku no kawaii imôto ga tsûhan de domazona omocha o kônyû shi makutte orimasu... shikamo daihiki de Muka...	Boku no kawaii imôto ga tsû
3	tt11002712	movie	Response Of God Of White Coat Natural Pure Nurse Always Seduces With Unprotected Chillisms Even	Response Of God Of White (
4	tt1269374	movie	A Bad Day at the Cemetery	La cripta de las mujeres mal
5	tt17372876	movie	All-American Cheerleaders Vol. 1	All-American Cheerleaders V
6	tt1883227	movie	Iniciando Parejas Vol. 4	Iniciando Parejas Vol. 4
7	tt3324654	movie	Bunga Bunga 3D	Bunga Bunga 3D
8	tt0132119	movie	L'ereditiera	L'ereditiera
9	tt19373584	movie	9 Nights	9 Nights
10	tt20225340	movie	If It Feels Good Vol. 3	If It Feels Good Vol. 3
11	tt10623034	movie	Kuroinu: Kedakaki Seijo wa Hakudaku ni Somaru	Kuroinu: Kedakaki Seijo wa I
12	tt15072388	movie	Big White Tits and Large Black Dicks 4	Big White Tits and Large Bla
13	tt15321622	movie	Piss Pigs	Piss Pigs
14	tt19895052	movie	Foreign Policy	Foreign Policy
15	tt21958162	movie	STARS	STARS
16	tt30745982	movie	Little Asians 6	Little Asians 6
17	tt19889428	movie	Fuck & Travel	Fuck & Travel
18	tt20216686	movie	Fuck My Super Tight Asshole	Fuck My Super Tight Asshol
19	tt2800092	movie	My Daughter Has a Dick 3	My Daughter Has a Dick 3
20	tt19391392	movie	Bachelorette Party	Bachelorette Party
21	tt15170096	movie	Cougars crave young kittens 3	Cougars crave young kitten

Total rows: 118 of 118 Query complete 00:00:00.056 Ln 5, Col 1

A clear decrease in execution time to 0.056 seconds can be observed in the above image for the same query after applying indexing. The analysis using the EXPLAIN tool is as follows:

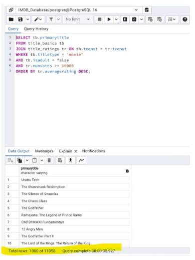


```
select * from title_basics where titletype = 'movie' and isadult = true and runtimeminutes >= 150;
```

```
Data Output   Messages   Explain ×   Notifications
Graphical   Analysis   Statistics
#    Node
1.   → Index Scan using idx_title_basics_movie_adult_runtime on title_basics as title_basics
       Index Cond: (((titletype)::text = 'movie'::text) AND (isadult = true) AND (runtimeminutes >= 150))
```

The picture provides an analysis of performance improvement, we can observe that the index we have created which is idx_title_basics_movie_adult_runtime is being utilized to retrieve the data.

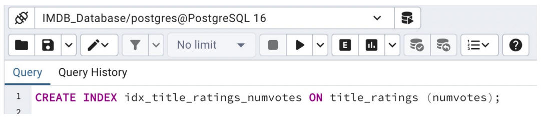
- **Problematic Query 2:**



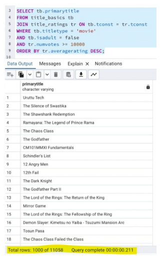
```
IMDB_Database/postgres@PostgreSQL 16
Query   Query History
SELECT tb.primarytitle
FROM title_basics tb
JOIN title_ratings tr ON tb.tconst = tr.tconst
WHERE tb.titletype = 'movie'
AND tb.isadult = false
AND tr.numvotes >= 10000
ORDER BY tr.averagerating DESC;
```

```
Data Output   Messages   Explain ×   Notifications
     primarytitle
     character varying
1    Uruttu Tech
2    The Shawshank Redemption
3    The Silence of Swastika
4    The Chaos Class
5    The Godfather
6    Ramayana: The Legend of Prince Rama
7    CM101MMXI Fundamentals
8    12 Angry Men
9    The Godfather Part II
10   The Lord of the Rings: The Return of the King
Total rows: 1000 of 11058     Query complete 00:00:05.927
```

As can be observed from the screenshot, the query is taking 5.927 seconds to fetch the data, which indicates slow performance. To reduce the execution time, we implemented indexing on 'numvotes' colum of title_ratings relaiton. This way, the data is not fetched in sequential manner and indexing are used instead.
we created the index as follows:



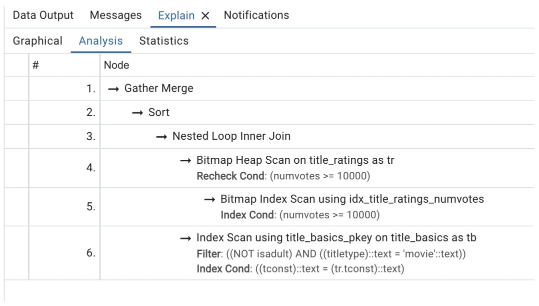
```
IMDB_Database/postgres@PostgreSQL 16
Query   Query History
CREATE INDEX idx_title_ratings_numvotes ON title_ratings (numvotes);
```



```
SELECT tb.primarytitle
FROM title_basics tb
JOIN title_ratings tr ON tb.tconst = tr.tconst
WHERE tb.titletype = 'movie'
AND tb.isadult = false
AND tr.numvotes >= 10000
ORDER BY tr.averagerating DESC;
```

```
Data Output   Messages   Explain ×   Notifications
     primarytitle
     character varying
1    Uruttu Tech
2    The Silence of Swastika
3    The Shawshank Redemption
4    Ramayana: The Legend of Prince Rama
5    The Chaos Class
6    The Godfather
7    CM101MMXI Fundamentals
8    Schindler's List
9    12 Angry Men
10   12th Fail
11   The Dark Knight
12   The Godfather Part II
13   The Lord of the Rings: The Return of the King
14   Mirror Game
15   The Lord of the Rings: The Fellowship of the Ring
16   Demon Slayer: Kimetsu no Yaiba - Tsuzumi Mansion Arc
17   Tosun Pasa
18   The Chaos Class Failed the Class
Total rows: 1000 of 11058     Query complete 00:00:00.211
```

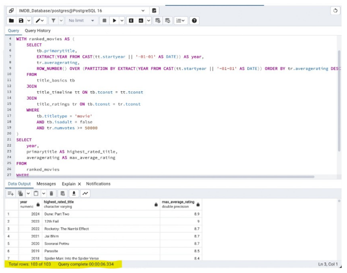
A clear decrease in execution time can be observed in the above image for the same query after creating indexing on title_basics relation. The analysis using the EXPLAIN tool is as follows:



```
Data Output   Messages   Explain ×   Notifications
Graphical   Analysis   Statistics
#    Node
1.   → Gather Merge
2.       → Sort
3.           → Nested Loop Inner Join
4.               → Bitmap Heap Scan on title_ratings as tr
                   Recheck Cond: (numvotes >= 10000)
5.                   → Bitmap Index Scan using idx_title_ratings_numvotes
                       Index Cond: (numvotes >= 10000)
6.               → Index Scan using title_basics_pkey on title_basics as tb
                   Filter: ((NOT isadult) AND ((titletype)::text = 'movie'::text))
                   Index Cond: ((tconst)::text = (tr.tconst)::text)
```

The picture provides an analysis of performance improvement, we can observe that the index we have created which is idx_title_ratings_numvotes is being utilized to retrieve the data.

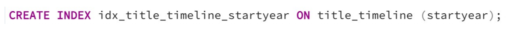
- **Problematic Query 3:**



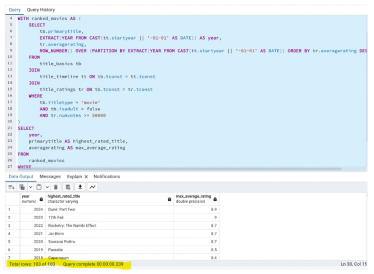
```
IMDB_Database/postgres@PostgreSQL 16
Query   Query History
WITH ranked_movies AS (
    SELECT
        tb.primarytitle,
        EXTRACT(YEAR FROM CAST(tt.startyear || '-01-01' AS DATE)) AS year,
        tr.averagerating,
        ROW_NUMBER() OVER (PARTITION BY EXTRACT(YEAR FROM CAST(tt.startyear || '-01-01' AS DATE)) ORDER BY tr.averagerating DESC
    FROM
        title_basics tb
    JOIN
        title_timeline tt ON tb.tconst = tt.tconst
    JOIN
        title_ratings tr ON tb.tconst = tr.tconst
    WHERE
        tb.titletype = 'movie'
        AND tb.isadult = false
        AND tr.numvotes >= 50000
)
SELECT
    year,
    primarytitle AS highest_rated_title,
    averagerating AS max_average_rating
FROM
    ranked_movies
WHERE
```

```
Data Output   Messages   Explain ×   Notifications
     year      highest_rated_title                  max_average_rating
     numeric   character varying                    double precision
1    2024      Dune: Part Two                       8.9
2    2023      12th Fail                            9
3    2022      Rocketry: The Nambi Effect           8.7
4    2021      Jai Bhim                             8.7
5    2020      Soorarai Pottru                      8.7
6    2019      Parasite                             8.5
7    2018      Spider-Man: Into the Spider-Verse    8.4
Total rows: 103 of 103     Query complete 00:00:06.334                 Ln 3, Col 1
```

As can be observed from the screenshot, the query is taking 6.334 seconds to fetch the data, which indicates slow performance. To reduce the execution time, we implemented indexing on 'startyear' column of title_timeline relation . This way, the data is not fetched in sequential manner and indexing are used instead.
we created the index as follows:



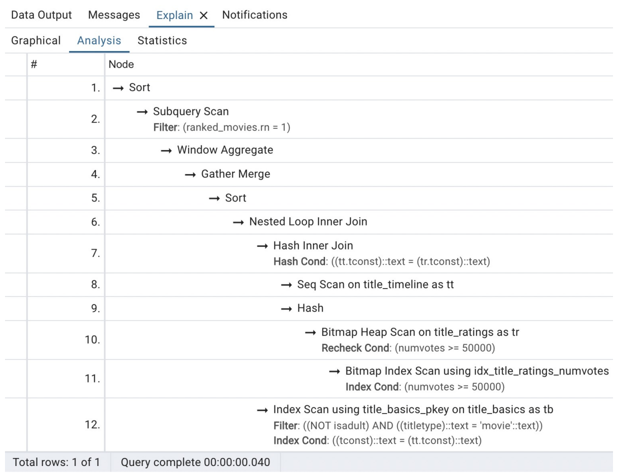
```
CREATE INDEX idx_title_timeline_startyear ON title_timeline (startyear);
```



```
Query   Query History
WITH ranked_movies AS (
    SELECT
        tb.primarytitle,
        EXTRACT(YEAR FROM CAST(tt.startyear || '-01-01' AS DATE)) AS year,
        tr.averagerating,
        ROW_NUMBER() OVER (PARTITION BY EXTRACT(YEAR FROM CAST(tt.startyear || '-01-01' AS DATE)) ORDER BY tr.averagerating DES
    FROM
        title_basics tb
    JOIN
        title_timeline tt ON tb.tconst = tt.tconst
    JOIN
        title_ratings tr ON tb.tconst = tr.tconst
    WHERE
        tb.titletype = 'movie'
        AND tb.isadult = false
        AND tr.numvotes >= 50000
)
SELECT
    year,
    primarytitle AS highest_rated_title,
    averagerating AS max_average_rating
FROM
    ranked_movies
WHERE
```

```
Data Output   Messages   Explain ×   Notifications
     year      highest_rated_title           max_average_rating
     numeric   character varying             double precision
1    2024      Dune: Part Two                8.9
2    2023      12th Fail                     9
3    2022      Rocketry: The Nambi Effect    8.7
4    2021      Jai Bhim                      8.7
5    2020      Soorarai Pottru               8.7
6    2019      Parasite                      8.5
7    2018      Capernaum                     8.4
Total rows: 103 of 103     Query complete 00:00:00.339                 Ln 30, Col 15
```

A clear decrease in execution time can be observed in the above image for the same query after creating indexing on title_basics relation. The analysis using the EXPLAIN tool is as follows:



```
#    Node
1.   → Sort
2.     → Subquery Scan
         Filter: (ranked_movies.rn = 1)
3.       → Window Aggregate
4.         → Gather Merge
5.           → Sort
6.             → Nested Loop Inner Join
7.               → Hash Inner Join
                   Hash Cond: ((tt.tconst)::text = (tr.tconst)::text)
8.                 → Seq Scan on title_timeline as tt
9.                 → Hash
10.                  → Bitmap Heap Scan on title_ratings as tr
                       Recheck Cond: (numvotes >= 50000)
11.                    → Bitmap Index Scan using idx_title_ratings_numvotes
                         Index Cond: (numvotes >= 50000)
12.              → Index Scan using title_basics_pkey on title_basics as tb
                   Filter: ((NOT isadult) AND ((titletype)::text = 'movie'::text))
                   Index Cond: ((tconst)::text = (tt.tconst)::text)
Total rows: 1 of 1    Query complete 00:00:00.040
```

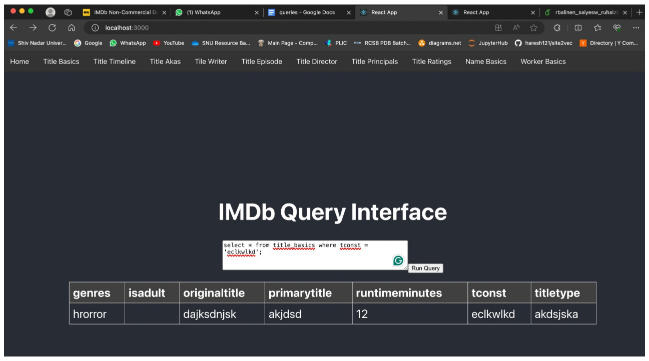
The picture provides an analysis of performance improvement, we can observe that the index we have created which is idx_title_timeline_startyear is being utilized, to retrieve the data.

X. Conclusion

To conclude, with the huge amount of data in entertainment area, it's very important to use a well-designed SQL database for making full use of IMDb's dataset. Such type of database gives a solid framework that helps in exploring the complex world of entertainment data and makes it easier to find useful information according to different needs from people involved in this industry.

XI. Website

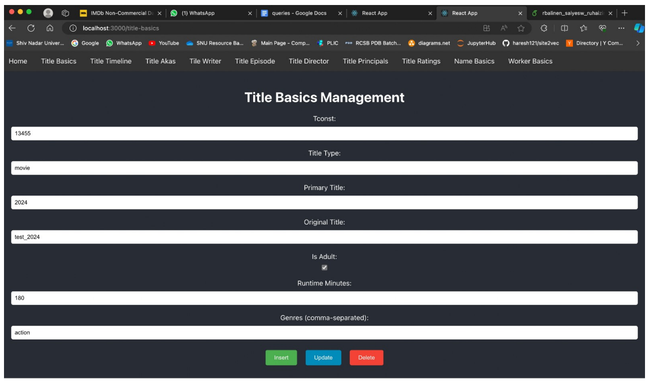
Following are some images of the website we created to access the database

- 

IMDb Query Interface

```
select * from title_basics where tconst =
'eclkwlkd';
```

genres	isadult	originaltitle	primarytitle	runtimeminutes	tconst	titletype
hrorror		dajksdnjsk	akjdsd	12	eclkwlkd	akdsjska

- 

Title Basics Management

Tconst:	13455
Title Type:	movie
Primary Title:	2024
Original Title:	test_2024
Is Adult:	✓
Runtime Minutes:	180
Genres (comma-separated):	action

XII. References

- https://www.imdb.com/interfaces/
- https://www.geeksforgeeks.org/boyce-codd-normal-form-bcnf/
- https://www.ibm.com/docs/en/informix-servers/14.10?topic=integrity-referential