IMDb Movie Database

1st Epuru Sai Muralidhar
Computer Science and Engineering
UBID : saimural
Buffalo, US
saimural@buffalo.edu

2nd Alla Leela Srija
Computer Science and Engineering
UBID : lalla
Buffalo, US
lalla@buffalo.edu

3rd Jampala Vishnu Teja
Computer Science and Engineering
UBID : vjampala
Buffalo, US
vjampala@buffalo.edu

***Abstract*—The entertainment industry has witnessed remarkable growth in the production of movies and TV shows, leading to a vast and ever-expanding collection of content. However, managing and accessing information about these works can be daunting, especially when relying on traditional methods such as spreadsheets(Excel) or paper-based records. This lack of a centralized and structured database system can result in inefficiencies, data inconsistencies, and difficulty retrieving relevant information.**

***Index Terms*—Database, Relations, Movies, Normal Forms**

I. Problem Statement

In the digital age, access to comprehensive and well-organized movie information has become essential for both entertainment enthusiasts and industry professionals alike. However, managing and maintaining accurate data for a vast collection of movies across various genres, languages, and eras can be a daunting task. Relying on spreadsheets or traditional file-based systems often leads to data redundancy, inconsistencies, and inefficient data retrieval processes.

A. Background of Problem

Previously, managing movie-related data was done manually, such as maintaining spreadsheets or physical records. This approach is time-consuming and error-prone and lacks the scalability and flexibility needed to handle the exponential growth of movie and TV show productions. Additionally, the absence of a centralized repository for movie data often leads to data silos, making it challenging to access and analyze information from multiple sources. The need for a comprehensive and structured database system for movies has become increasingly important as the entertainment industry expands. Stakeholders, including film enthusiasts, critics, researchers, and industry professionals, require efficient access to accurate and up-to-date information about movies, cast and crew members, ratings, and other relevant details.

Addressing this challenge is significant because it can streamline data management processes, improve data accessibility, and enable more informed decision-making within the entertainment industry.

B. Project's Contribution

Our IMDb Movie Database project aims to leverage the comprehensive IMDb dataset, which contains a wealth of information about movies, TV shows, cast and crew, ratings, and user reviews. By creating a robust and well-designed database system, we can contribute to the movie industry and related domains in the following ways:

- **Centralized Data Repository**: Our database will serve as a centralized repository for movie-related data, consolidating information from various sources into a single, structured location. This will eliminate data silos and ensure consistent and reliable access to movie information.
- **Data Integrity and Consistency**: By implementing proper database design principles and data validation mechanisms, our system will ensure data integrity and consistency, reducing the risk of errors and inconsistencies that can arise from manual data handling processes.
- **Efficient Data Retrieval and Analysis**: With a well-structured database, querying and retrieving specific information about movies, cast and crew, genres, ratings, and other relevant details will become more efficient. This will enable stakeholders to perform in-depth analyses and generate valuable insights.
- **Scalability and Flexibility**: Our database system will be designed with scalability in mind, allowing for the seamless integration of new data sources and the ability to accommodate the growing volume of movie and TV show productions.
- **Data Accessibility**: By providing a user-friendly interface or API, our database will facilitate easy access to movie-related data for various stakeholders, including film enthusiasts, researchers, industry professionals, and application developers.

II. Target User

The IMDb Movie Database Management System will cater to a diverse range of users, including movie enthusiasts, industry professionals, researchers, and organizations within the entertainment industry. The target user groups can be categorized as follows:

- *Movie Enthusiasts and General Audience*: Casual movie fans who seek comprehensive information about their favorite films, actors, directors, and genres.
- *Industry Professionals*: Filmmakers, directors, producers, and screenwriters seek information on past productions, cast and crew details, and box office performance for research and reference purposes.
- *Research and Academic Institutions*: Film studies researchers and scholars are investigating various aspects

of the movie industry, such as trends, cultural influences, and historical perspectives.

- *Organizations and Businesses*: Streaming platforms and video-on-demand services utilizing the database to enhance their content libraries and provide accurate movie information to subscribers. Movie-related websites, blogs, and online communities seeking reliable and up-to-date information to share with their audience.
- *Database Administrators (DBAs)*: Responsible for the database system's overall management, performance optimization, and security. Implement backup and recovery strategies, monitor database performance, and handle database upgrades and migrations.
- *Data Analysts and Quality Assurance Team*: Perform regular data quality checks, identify and resolve any data inconsistencies or errors. Analyze data trends, usage patterns, and user feedback to improve the database structure and enhance the user experience.
- *Security and Compliance Team*: Implement robust security measures, such as data encryption, access controls, and auditing mechanisms, to protect the confidentiality and integrity of the movie data. Ensure compliance with relevant data protection regulations, such as the General Data Protection Regulation (GDPR) or industry-specific standards. Conduct regular security audits and vulnerability assessments to identify and mitigate potential risks.

III. RELATIONS

- **basics_imdb** (tconst, titleType, primaryTitle, originalTitle, isAdult, startYear, endYear, runtimeMinutes, genres)
  Primary Key: tconst is the primary key as it uniquely identifies the row in basics_imdb table.
  1) tconst (Varchar(30) PRIMARY KEY NOT NULL): An alphanumeric unique identifier of the title
  2) titleType (varchar(100)): The type/format of the title
  3) primaryTitle (varchar(600)): The more popular title / the title used by the filmmakers on promotional materials at the point of release
  4) originalTitle (vachar(600)): Original title, in the original language
  5) isAdult (varchar(20)): 0: non-adult title; 1: adult title
  6) startYear (int): represents the release year of a title. In the case of TV Series, it is the series start year
  7) endYear (int): TV Series end year. '\N' for all other title types
  8) runtimeMinutes (int): primary runtime of the title, in minutes
  9) genres (varchar[]): includes up to three genres associated with the title
- **names_imdb** (nconst, primaryName, birthYear, deathYear, primaryProfession, knownForTitles)
  Primary Key: nconst is the primary key as it uniquely identifies the row in names_imdb table.
  1) nconst(varchar(20) PRIMARY KEY NOT NULL) - alphanumeric unique identifier of the name/person
  2) primaryName(varchar(500)) - name by which the person is most often credited
  3) birthYear(int) - in YYYY format
  4) deathYear(int) - in YYYY format if applicable, else '\N'
  5) primaryProfession(varchar[]) - the top-3 professions of the person
  6) knownforTitles (varchar[]) - titles the person is known for
- **akas_imdb** (titleId, ordering, title, region, language, types, attributes,isOriginalTitle)
  Primary Key: titleId , Ordering are the primary keys as they together uniquely identifies a row in akas_imdb table.
  Foreign Key: titleId is the foreign key - reference from basics_imdb (tconst) table.
  1) titleId(varchar(10) NOT NULL): An alphanumeric unique identifier of the title referencing basic_imdb UPDATE/DELETE restricted
  2) ordering(int NOT NULL): A number to uniquely identify rows for a given titleId
  3) title(varchar(600)): The localized title
  4) region(varchar(10)): The region for this version of the title
  5) language(varchar(60)): The language of the title
  6) types(varchar(600)): Enumerated set of attributes for this alternative title. One or more of the following: "alternative", "dvd", "festival", "tv", "video", "working", "original", "imdbDisplay". New values may be added in the future without warning. For this we are taking the first element.
  7) attributes(varchar(600)): Additional terms to describe this alternative title, not enumerated. Considering by taking the first element.
  8) isOriginalTitle(int): 0: not original title; 1: original title
- **crew_imdb** (tconst, directors, writers)
  Primary Key: tconst is the primary key as it uniquely identifies the row in crew_imdb table.
  Foreign Key: tconst is the foreign key - reference from basics_imdb (tconst) table.
  1) tconst(varchar(30) NOT NULL): alphanumeric unique identifier of the title UPDATE/DELETE Restricted
  2) directors(varchar[]): director(s) of the given title
  3) writers(varchar[]): writer(s) of the given title
- **episode_imdb**(tconst, parentTconst, seasonNumber, episodeNumber)
  Primary Key: tconst is the primary key as it uniquely identifies the row in episode_imdb table.
  Foreign Key: parentTconst is the foreign key - reference from basics_imdb (tconst) table, tconst is the foreign key - reference from basics_imdb (tconst) table
  1) tconst(varchar(20) NOT NULL): alphanumeric identifier of episode

2) parentTconst(varchar(20)): alphanumeric identifier of the parent TV Series. UPDATE/DELETE Restricted.
3) seasonNumber(int): season number the episode belongs to
4) episodeNumber(int):episode number of the tconst in the TV series

- **principals_imdb** (tconst, ordering, nconst, job, characters)
  Primary Key: tconst, ordering are the primary keys as they together uniquely identifies the row in principals_imdb table.
  Foreign Key: nconst is the foreign key - reference from names_imdb (tconst) table, tconst is the foreign key - reference from basics_imdb (tconst) table
  1) tconst(varchar(20) NOT NULL): alphanumeric unique identifier of the title
  2) ordering(int NOT NULL): a number to uniquely identify rows for a given titleId
  3) nconst(varchar(20)): alphanumeric unique identifier of the name/person. UPDATE/DELETE Restricted.
  4) category(varchar):the category of job that person was in
  5) job(varchar): the specific job title if applicable, else '\N'
  6) characters(varchar): the name of the character played if applicable, else '\N'
- **ratings_imdb** (tconst, averageRating, numVotes)
  Primary Key: tconst is the primary key as it uniquely identifies the row in ratings_imdb table.
  Foreign Key: tconst is the foreign key - reference from basics_imdb (tconst) table.
  1) tconst(varchar(30) NOT NULL): alphanumeric unique identifier of the title UPDATE/DELETE Restricted.
  2) averageRating(double precision): weighted average of all the individual user ratings
  3) numVotes(int): number of votes the title has received

IV. BCNF

- **basics_imdb** table:
  We have removed genres column which is String Array and violates 1NF hence violating BCNF and created a new table **genres_imdb** for it as part of normalization.
  Primary Key: tconst
  All non-key attributes (titleType, primaryTitle, originalTitle, isAdult, startYear, endYear, runtimeMinutes) are fully functionally dependent on the primary key tconst. This table is in BCNF.
- **names_imdb** table:
  We have removed primaryProfession, knownForTitles column which are of type String Arrays and violates 1NF hence violating BCNF. Created new tables for **primaryprofession_imdb** and **knownForTitles_imdb** respectively as part of normalization.
  Primary Key: nconst
  All non-key attributes (primaryName, birthYear, deathYear, knownForTitles) are fully functionally dependent on the primary key nconst. This table is in BCNF.
- **akas_imdb** table:
  Composite Primary Key: titleId, ordering
  All non-key attributes (title, region, language, types, attributes, isOriginalTitle) are fully functionally dependent on the composite primary key (titleId, ordering). This table is in BCNF.
- **episode_imdb** table:
  Primary Key: tconst
  Non-key attributes (parentTconst, seasonNumber, episodeNumber) are fully functionally dependent on the primary key tconst. This table is in BCNF.
- **principals_imdb** table:
  Composite Primary Key: tconst, ordering
  All non-key attributes (nconst, category, job, characters) are fully functionally dependent on the composite primary key (tconst, ordering). This table is in BCNF.
- **ratings_imdb** table:
  Primary Key: tconst
  Non-key attributes (averageRating, numVotes) are fully functionally dependent on the primary key tconst. This table is in BCNF.
- **genres_imdb** table:
  Primary Key: tconst, genre
  All attributes in the table (tconst, genre) are either part of the primary key or fully functionally dependent on the primary key. This table is in BCNF. This table is in BCNF.
- The **crew_imdb** table violates BCNF as the attributes directors and writers are multivalued attributes.
  - **directors_imdb** table:
    Primary Key: tconst, nconst
    All attributes in the table (tconst, nconst) are either part of the primary key or fully functionally dependent on the primary key. This table is in BCNF.
  - **writers_imdb** table:
    Primary Key: tconst, nconst
    All attributes in the table (tconst, nconst) are either part of the primary key or fully functionally dependent on the primary key. This table is in BCNF.
- **primaryprofession_imdb** table:
  Primary Key: nconst, primaryprofession
  All attributes in the table (nconst, primaryprofession) are either part of the primary key or fully functionally dependent on the primary key. This table is in BCNF.
- **knownForTitles_imdb** table:
  Primary Key: nconst, knownForTitle
  All attributes in the table (nconst, knownForTitle) are either part of the primary key or fully functionally dependent on the primary key. This table is in BCNF.

V. E/R Diagram

The Entity-Relationship Diagram (ERD) in the Figure 1 outlines the main entities and their relationships within the database before normalization and Figure 2 shows the ERD after normalization.

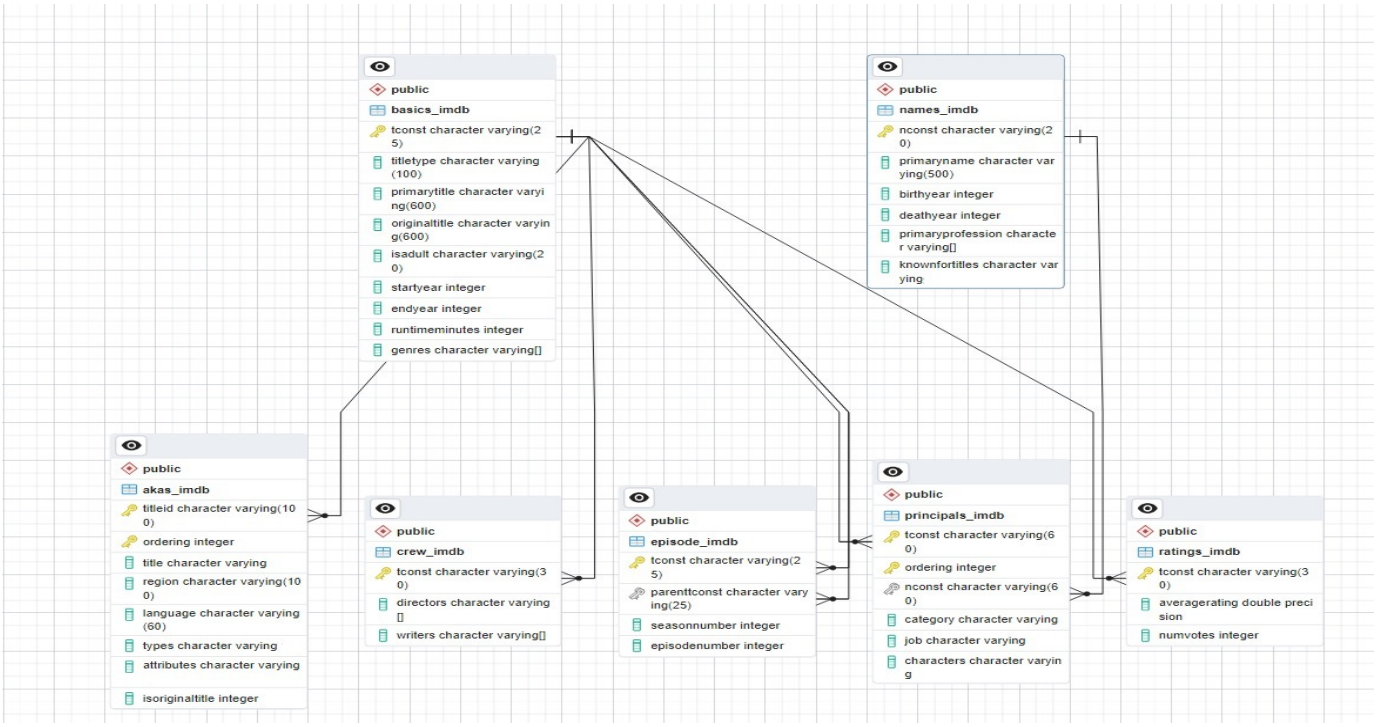


Fig. 1. E/R Diagram before BCNF

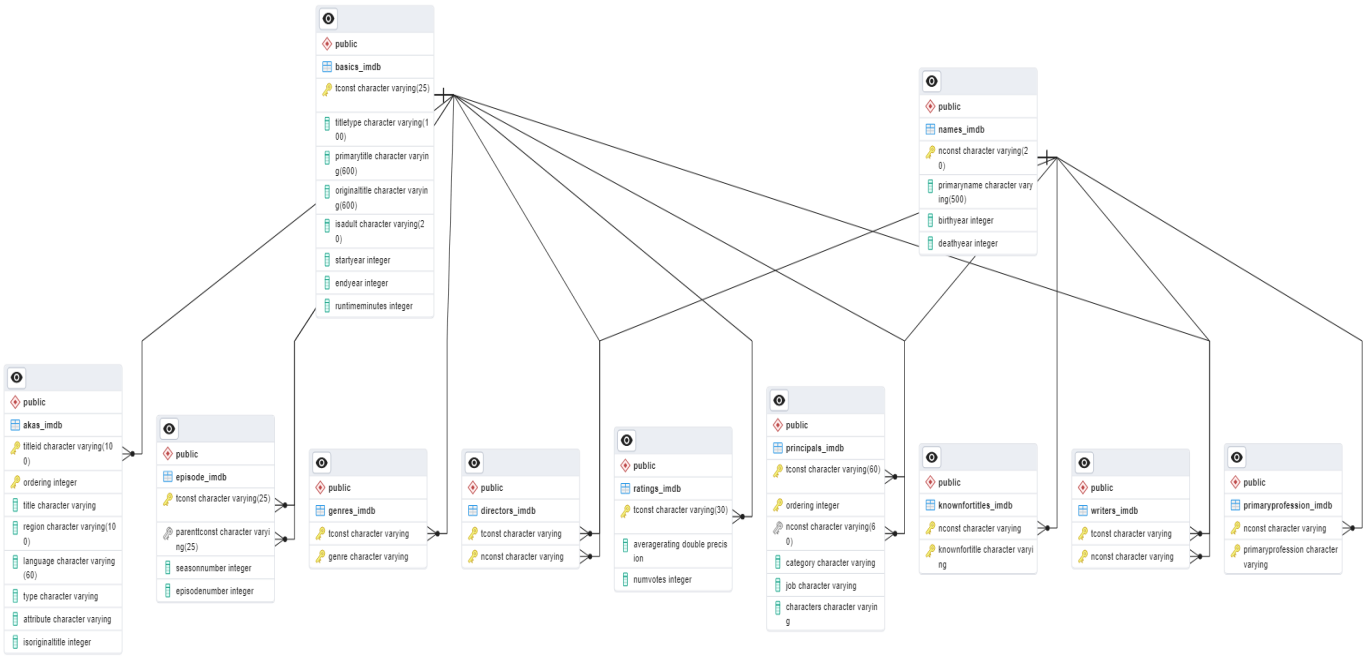


Fig. 2. E/R Diagram after BCNF

VI. Problems with Dataset

A. Loading Dataset

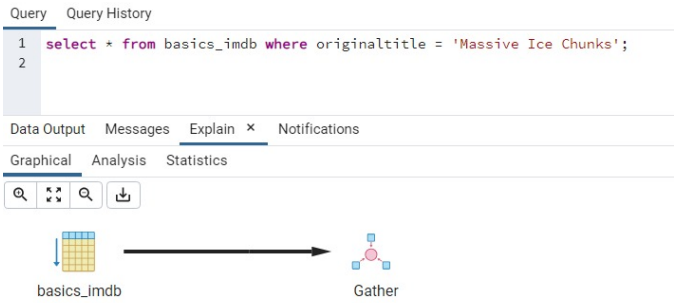
While loading the dataset, we faced a few issues due to inconsistencies in data and dataset size.

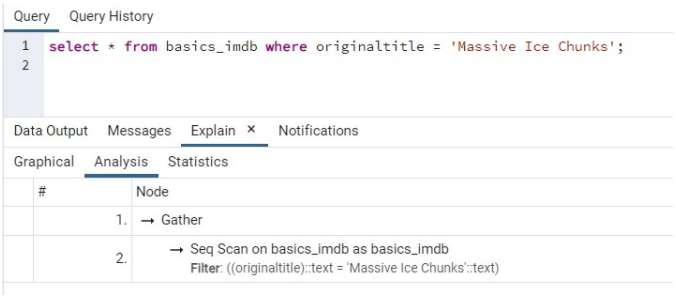
1) Large dataset Size: We had already identified and eliminated redundant data in our large dataset, which was time-consuming and complex due to multiple dependencies and functional relationships. Despite the challenges involved in dealing with such intricacies, we successfully addressed the issue of data redundancy, mitigating potential inconsistencies and anomalies during the normalization process.

While dividing the dataset, we had to divide the table entries into chunks so as to get the new table due to its large number.

2) Data inconsistencies: There were few entries in **crew_imdb**, **names_imdb** and **principals_imdb** that have foreign keys referenced from **basics_imdb**. To handle the inconsistencies, we had to drop some entries from these tables with *tconst* values that are not in **basics_imdb**. This is something that cannot be avoided in order to stick to the SQL constraints. The query used to remove the entries was
DELETE FROM principals_imdb WHERE NOT EXISTS (SELECT 1 FROM basics_imdb WHERE basics_imdb.tconst = principals_imdb.tconst);
ALTER TABLE principals_imdb ADD CONSTRAINT principals_tconst_fkey FOREIGN KEY (tconst) REFERENCES basics_imdb (tconst)

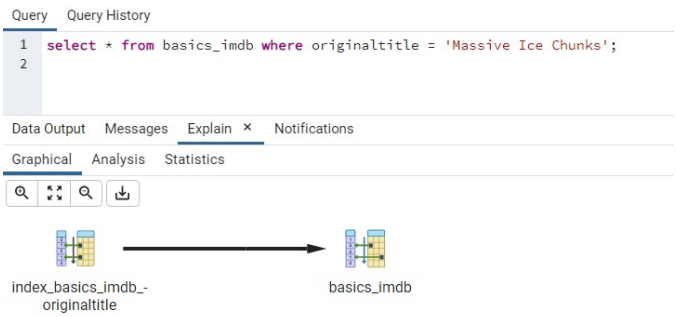
3) Slower Data Retrieval: For queries that require scanning the entire table, the execution time is significantly higher due to the large size of the dataset. However, indexing can benefit queries that exhibit prolonged execution times caused by extensive table-searching operations. By indexing the relevant columns, the time required to locate and retrieve the necessary data can be substantially reduced, enhancing the overall query performance.
For example for the query - select * from basics_imdb where originaltitle = 'Massive Ice Chunks';

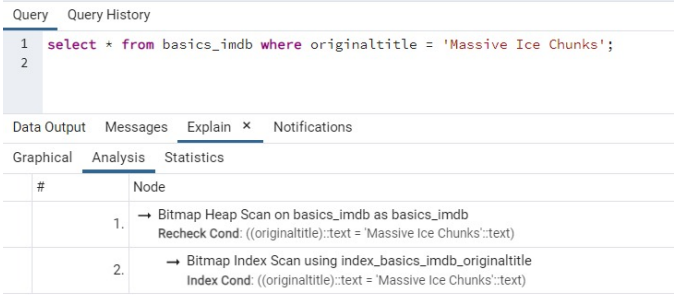




In Figures VI-A3 and VI-A3, we can see the execution plan lists the steps the SQL engine took to execute the query. It mentions a "Gather" step, indicating data collection from potentially multiple sources or partitions, followed by a "Sequential Scan" on the basics_imdb table. The filter condition applied is that originaltitle must exactly match 'Massive Ice Chunks'. This suggests that no index is used for this query; instead, a full table scan is performed.

On adding indexing to the column - CREATE INDEX idx_basics_originaltitle ON public.basics_imdb(originaltitle);



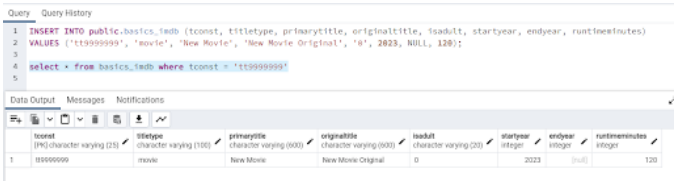


In Figures VI-A3 and VI-A3, we can see the execution plan lists the steps taken by the SQL engine to execute the query using indexing. The SQL engine performs a Bitmap Heap Scan on the basics_imdb table. This step involves scanning the table rows pointed to by a bitmap constructed from an index scan. Before the heap scan, a Bitmap Index Scan uses the index index_basics_imdb_originaltitle. The index condition checked is originaltitle = 'Massive Ice Chunks', which filters the index entries to find rows that meet the query criteria. The textual

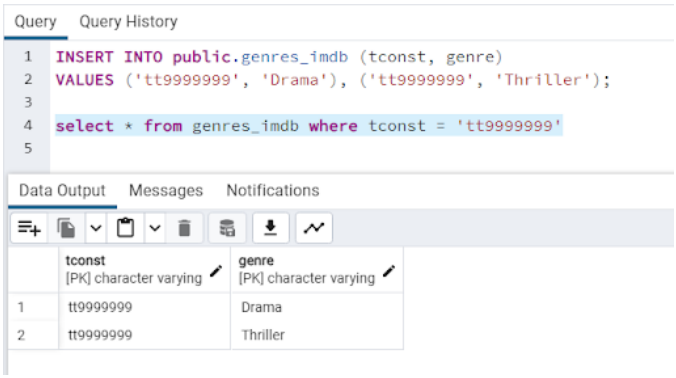
plan shows a hierarchical operation where the index is used first to efficiently locate rows in the database that match the query condition, reducing the number of rows the SQL engine must scan directly in the table.

VII. Queries

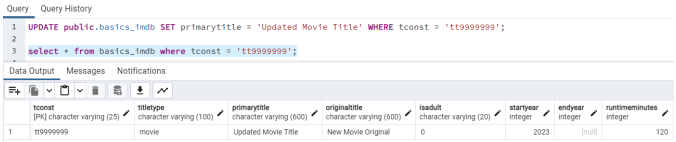
1) Insert a new title into basics_imdb
Query: INSERT INTO public.basics_imdb (tconst, titletype, primarytitle, originaltitle, isadult, startyear, endyear, runtimeminutes) VALUES (’tt9999999’, ’movie’, ’New Movie’, ’New Movie Original’, ’0’, 2023, NULL, 120);



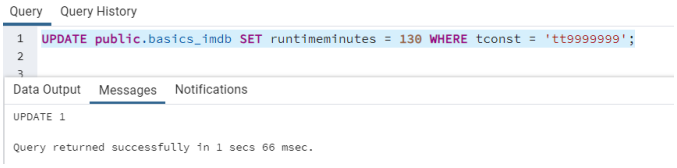
2) Insert multiple genres for a title in genres_imdb
Query: INSERT INTO public.genres_imdb (tconst, genre) VALUES (’tt9999999’, ’Drama’), (’tt9999999’, ’Thriller’);

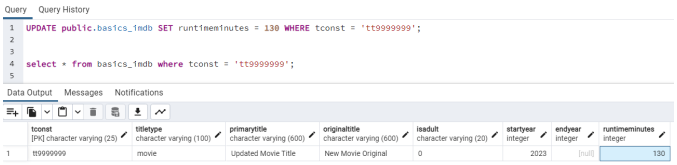


3) Update the title of a movie in basics_imdb
Query: UPDATE public.basics_imdb SET primarytitle = ’Updated Movie Title’ WHERE tconst = ’tt9999999’;

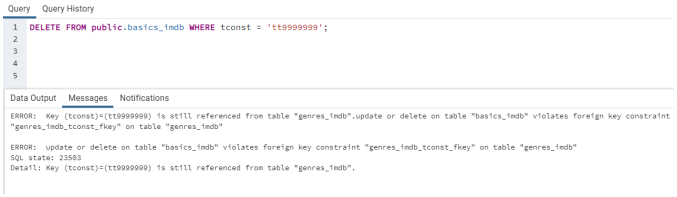


4) Update the run time of a movie in basics_imdb:
Query: UPDATE public.basics_imdb SET runtimeminutes = 130 WHERE tconst = ’tt9999999’;





5) Delete a title from basics_imdb
Query: DELETE FROM public.basics_imdb WHERE tconst = ’tt9999999’;



The above query does not work because of foreign key constraint.

6) Delete a genre entry from genres_imdb
Query: DELETE FROM public.genres_imdb WHERE tconst = ’tt9999999’;

Query Query History

```
1  DELETE FROM public.basics_imdb WHERE tconst = 'tt9999999';
2
3
```

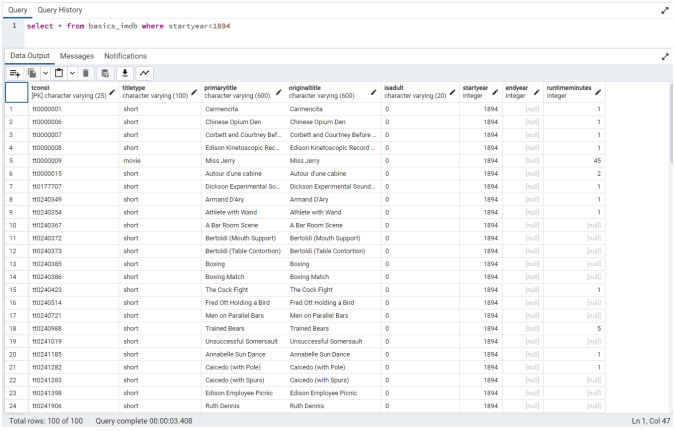
Data Output Messages Notifications

```
DELETE 1

Query returned successfully in 236 msec.
```

Now it can be deleted from basics_imdb too because there is no foreign key in genres_imdb table.

7) Select all information about a movie
Query: SELECT * FROM public.basics_imdb WHERE startyear = 1894;



8) Join basics_imdb with ratings_imdb to get titles and their ratings
Query: SELECT b.primarytitle, r.averagerating FROM public.basics_imdb b JOIN public.ratings_imdb r ON b.tconst = r.tconst ORDER BY r.averagerating DESC LIMIT 10;

Query Query History

```
1  SELECT b.primarytitle, r.averagerating
2  FROM public.basics_imdb b
3  JOIN public.ratings_imdb r ON b.tconst = r.tconst
4  ORDER BY r.averagerating DESC LIMIT 10;
5
```

Data Output Messages Notifications

	primarytitle character varying (600)	averagerating double precision
1	Koll	10
2	Eine große Familie	10
3	Sechs Personen suchen einen Autor	10
4	A View of Bosnia	10
5	Renegades 2	10
6	Mymo's Adventures	10
7	Por orden de aparición	10
8	Episode #1.1	10
9	Cekor Pred Vremeto	10
10	Grapes	10

9) Use a GROUP BY to count the number of titles in each genre
Query: SELECT genre, COUNT(tconst) AS title_count FROM public.genres_imdb GROUP BY genre;

```
SELECT genre, COUNT(tconst) AS title_count
FROM public.genres_imdb
GROUP BY genre;
```

	genre character varying	title_count bigint
1		359
2	Action	440420
3	Adult	338039
4	Adventure	421132
5	Animation	540740
6	Biography	115836
7	Comedy	2119030
8	Crime	448598
9	Documentary	1024712
10	Drama	3064561
11	Family	793850
12	Fantasy	223404
13	Film-Noir	883
14	Game-Show	395236
15	History	159218
16	Horror	192737
17	Music	403128
18	Musical	90461
19	Mystery	217957
20	News	962119
21	Reality-TV	605469
22	Romance	1001447

Total rows: 29 of 29 Query complete 00:00:05.805

10) Select using a subquery to find directors of a specific movie
Query: SELECT n.primaryname FROM public.names_imdb n WHERE n.nconst IN (SELECT d.nconst FROM public.directors_imdb d WHERE d.tconst = 'tt9999999');

```
SELECT n.primaryname
FROM public.names_imdb n
WHERE n.nconst IN (SELECT d.nconst FROM public.directors_imdb d WHERE d.tconst = 'tt0000009');
```

	primaryname character varying (500)
1	Alexander Black

11) List all genres and their movies using LEFT JOIN
Query: SELECT g.genre, b.primarytitle FROM public.genres_imdb g LEFT JOIN public.basics_imdb b ON g.tconst = b.tconst;

```
SELECT g.genre, b.primarytitle
FROM public.genres_imdb g
LEFT JOIN public.basics_imdb b ON g.tconst = b.tconst;
```

	genre character varying	primarytitle character varying (600)
1	Drama	My Friend Robinson
2	Comedy	The Girl on a Broomstick
3	Family	The Girl on a Broomstick
4	Fantasy	The Girl on a Broomstick
5	Adventure	The New Scooby-Doo Movies
6	Animation	The New Scooby-Doo Movies
7	Comedy	The New Scooby-Doo Movies
8	Comedy	The Wacky World of Jonathan Winters
9	Comedy	Girls on the Road
10	Drama	Girls on the Road
11	Thriller	Girls on the Road
12	Action	The Jerusalem File
13	Drama	The Jerusalem File
14	Thriller	The Jerusalem File
15	Drama	Lady Caroline Lamb
16	History	Lady Caroline Lamb
17	Romance	Lady Caroline Lamb
18	Drama	A magyar ugaron
19	Drama	Morbidness
20	Thriller	Morbidness
21	Drama	Poslizg

Total rows: 1000 of 16669996 Query complete 00:00:56.389

12) Select titles and directors, ordering by the start year of the title
Query: SELECT b.primarytitle, n.primaryname, b.startyear FROM public.basics_imdb b JOIN public.directors_imdb d ON b.tconst = d.tconst JOIN public.names_imdb n ON d.nconst = n.nconst ORDER BY b.startyear DESC;

Query Query History

```
SELECT b.primarytitle, n.primaryname, b.startyear
FROM public.basics_imdb b
JOIN public.directors_imdb d ON b.tconst = d.tconst
JOIN public.names_imdb n ON d.nconst = n.nconst
ORDER BY b.startyear DESC LIMIT 100;
```

Data Output Messages Notifications

	primarytitle character varying (600)	primaryname character varying (500)	startyear integer
1	Episode #1.171	Tekin Akmansoy	[null]
2	Episode #1.284	Sujeet Kumar	[null]
3	Episode #1.171	Arzu Akmansoy	[null]
4	Episode #1.197	Rajesh Radheylal Gupta	[null]
5	Episode #1.137	Arzu Akmansoy	[null]
6	Episode #1.284	Pawan Kumar	[null]
7	Episode #1.95	Ram Narayan Uragonda	[null]
8	Episode #1.153	Ucik Supra	[null]
9	Episode #1.556	Rajesh Radheylal Gupta	[null]
10	Episode #1.137	Tekin Akmansoy	[null]
11	Episode #1.1122	Keshab Malick	[null]
12	Episode #1.1619	Keshab Malick	[null]
13	Episode #1.663	Ucik Supra	[null]
14	Episode #1.104	Mohit Jha	[null]
15	Episode #1.105	Krishna Kant Chauhan	[null]
16	Episode #1.1303	Anil v Kumar	[null]
17	A Twisted Trade	Ajay Veermal	[null]
18	The Drifters	Dimo Petkov	[null]
19	Episode #1.928	Ucik Supra	[null]
20	Episode #1.188	Gita Asmara	[null]
21	15/30/40	Oksana Belousova	[null]

Total rows: 100 of 100 Query complete 00:04:09.821

13) Movies by Average Rating with Genre Filtering
Query: SELECT b.primarytitle, AVG(r.averagerating) AS AverageRating FROM public.basics_imdb b JOIN public.ratings_imdb r ON b.tconst = r.tconst JOIN public.genres_imdb g ON b.tconst = g.tconst WHERE g.genre = 'Drama' GROUP BY b.primarytitle HAVING COUNT(b.tconst) > 1 ORDER BY AverageRating DESC;

Query Query History

```
SELECT b.primarytitle, AVG(r.averagerating) AS AverageRating
FROM public.basics_imdb b
JOIN public.ratings_imdb r ON b.tconst = r.tconst
JOIN public.genres_imdb g ON b.tconst = g.tconst
WHERE g.genre = 'Drama'
GROUP BY b.primarytitle
HAVING COUNT(b.tconst) > 1
ORDER BY AverageRating DESC;
```

Data Output Messages Notifications

	primarytitle character varying (600)	averagerating double precision
1	Proof Against Kavya	10
2	Mi ultimo amigo	10
3	Washed Up	9.8
4	Triple A	9.7
5	Dev and Sonakshi get into a fight	9.649999999999999
6	Sanctified	9.600000000000001
7	Over & Over	9.600000000000001
8	Un dia cualquiera	9.6
9	Shin	9.6
10	Episode dated 12 September 2003	9.6
11	Tendlya	9.6
12	Das Paradies	9.6
13	Contradiction	9.55
14	House of Forever	9.55
15	Fast ein Poet	9.55
16	The Subway	9.55
17	Years	9.55
18	Episode dated 18 September 2000	9.5
19	EP 1	9.5

Total rows: 1000 of 36045 Query complete 00:00:20.596

14) Episodes Per Season Analysis
Query: SELECT e.parenttconst, e.seasonnumber, COUNT(e.episodenumber) AS EpisodesCount FROM public.episode_imdb e WHERE e.parenttconst = 'tt1234567' GROUP BY e.parenttconst, e.seasonnumber ORDER BY e.seasonnumber;

Query Query History

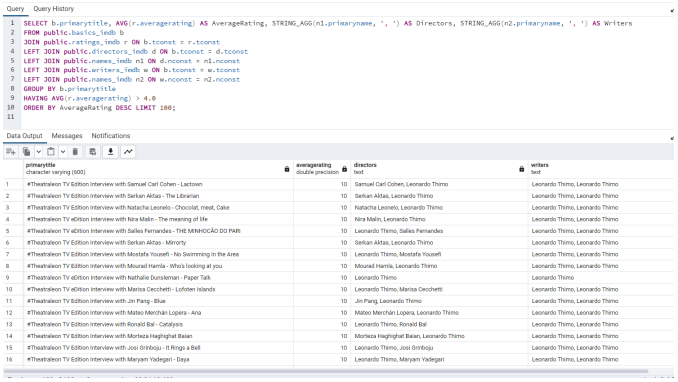
```
SELECT e.parenttconst, e.seasonnumber, COUNT(e.episodenumber) AS EpisodesCount
FROM public.episode_imdb e
WHERE e.parenttconst = 'tt0041038'
GROUP BY e.parenttconst, e.seasonnumber
ORDER BY e.seasonnumber;
```

Data Output Messages Notifications

	parenttconst character varying (25)	seasonnumber integer	episodescount bigint
1	tt0041038	1	52
2	tt0041038	2	26
3	tt0041038	3	52
4	tt0041038	4	52
5	tt0041038	5	39

15) Top Rated Movies with Directors and Writers
Query: SELECT b.primarytitle, AVG(r.averagerating) AS AverageRating, STRING_AGG(n1.primaryname, ', ') AS Directors, STRING_AGG(n2.primaryname, ', ') AS Writers FROM public.basics_imdb b JOIN public.ratings_imdb r ON b.tconst = r.tconst LEFT JOIN public.directors_imdb d ON b.tconst = d.tconst

LEFT JOIN public.names_imdb n1 ON d.nconst = n1.nconst LEFT JOIN public.writers_imdb w ON b.tconst = w.tconst LEFT JOIN public.names_imdb n2 ON w.nconst = n2.nconst GROUP BY b.primarytitle HAVING AVG(r.averagerating) > 8.0 ORDER BY AverageRating DESC;

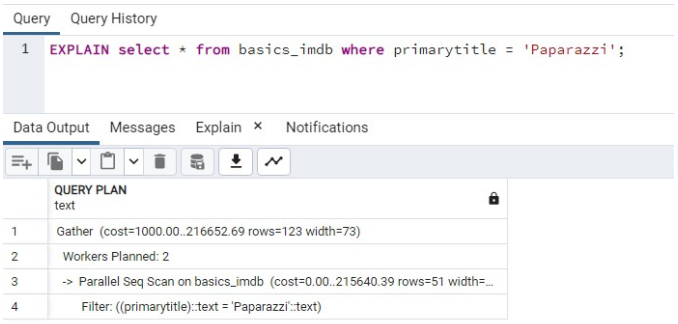


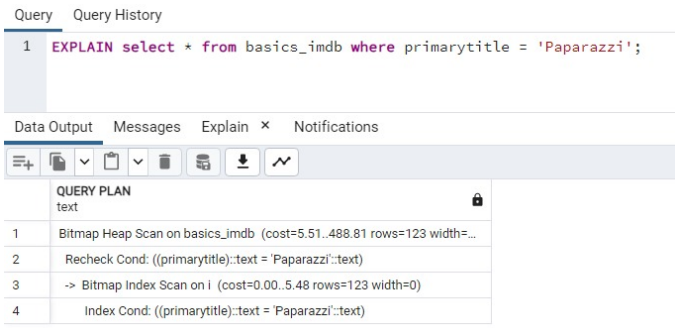
VIII. EXECUTION ANALYSIS

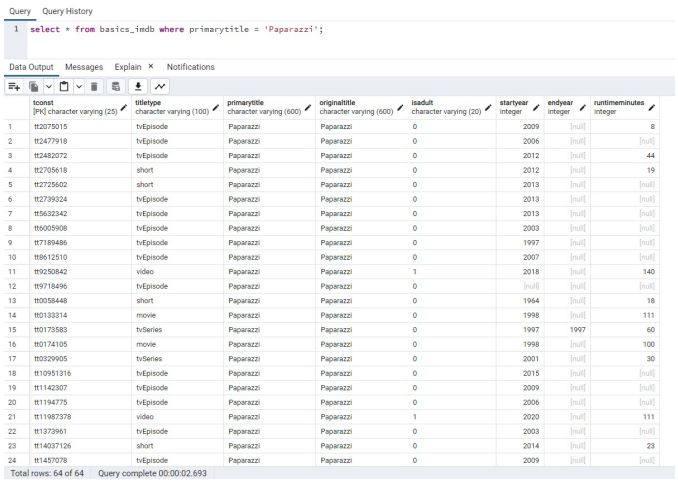
Working with large datasets often presents challenges in terms of query performance and execution time. As the volume of data grows, inefficient queries can significantly impact the overall system performance, leading to slow response times and potential bottlenecks. In this task, we will focus on analyzing query execution and identifying problematic queries that exhibit high costs or long execution times.

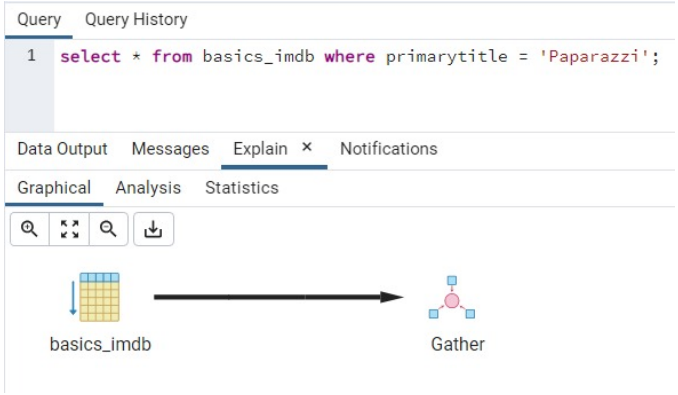
We will take some queries and analyse the execution times and costs with the help of *EXPLAIN* tool in Postgres.

- **select * from basics_imdb where primarytitle = 'Paparazzi';**
  Before indexing -









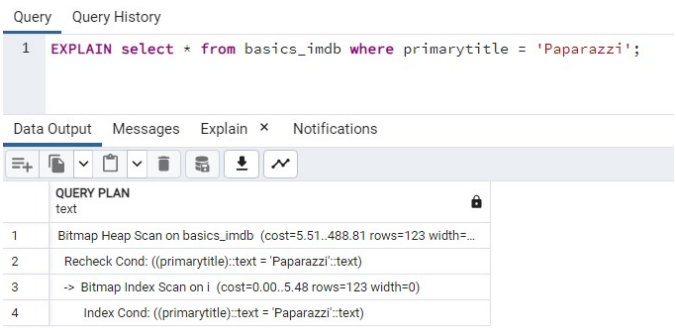
– **Cost:** 1000.00..216652.69 are the cost metrics associated with the operation. The cost value in PostgreSQL has two components:
Startup cost (1000.00): The cost incurred before the first row can be returned.
Total cost (216652.69): The estimated total cost to complete the execution of the node.

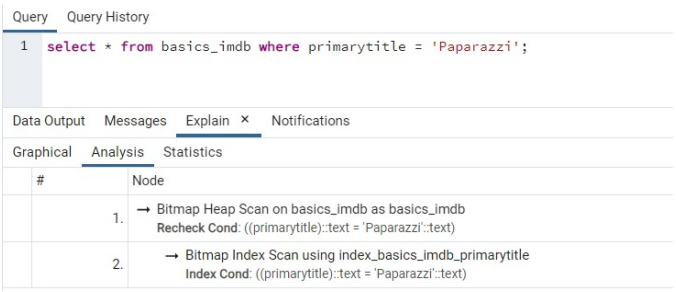
– *Parallel Seq Scan*: This indicates that the database will perform a sequential scan across multiple workers. A sequential scan is when the database reads

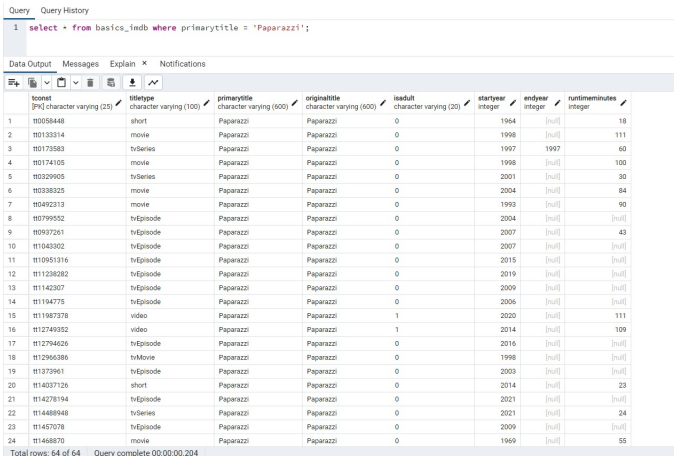
through every row in the table to find those that match the filter condition.

- *Execution Time*:2.693 seconds

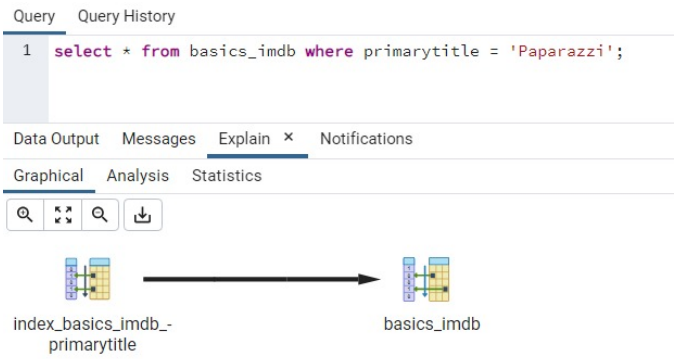
After Indexing -



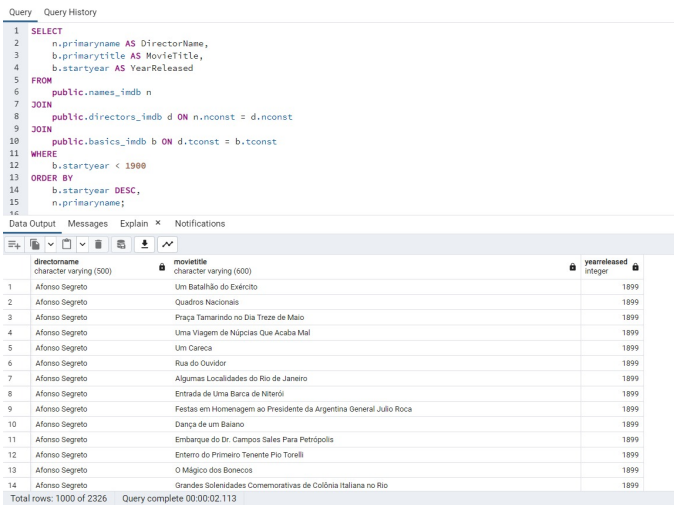




- *Bitmap Index Scan on index_basics_imdb_primarytitle* This part of the plan indicates that PostgreSQL is using a bitmap index scan on the index created for the primarytitle column. This is an efficient way to retrieve rows based on index keys that match the filter condition (primarytitle = 'Paparazzi'). The index scan is used to build a bitmap (a set of pointers) of the row locations in the table that match the criteria.
- *Bitmap Heap Scan on basics_imdb* After the bitmap index scan, a bitmap heap scan is performed. This step involves visiting the actual table rows pointed to by the bitmap created in the previous step. This is necessary to retrieve the full rows and ensure they meet all the query conditions.
- *Startup Cost:* 5.51
- *Total Cost:* 488.81
- *Bitmap Index Scan on index_basics_imdb_primarytitle*
  Cost: The cost of performing the index scan (0.00..5.48), which is quite low, indicating that using the index is efficient. *Rows:* Estimated number of rows that the index believes will match the condition (123).
- *Execution Time*:0.204 seconds



- **SELECT n.primaryname AS DirectorName, b.primarytitle AS MovieTitle, b.startyear AS YearReleased FROM public.names_imdb n JOIN public.directors_imdb d ON n.nconst = d.nconst JOIN public.basics_imdb b ON d.tconst = b.tconst WHERE b.startyear < 1900 ORDER BY b.startyear DESC, n.primaryname;**

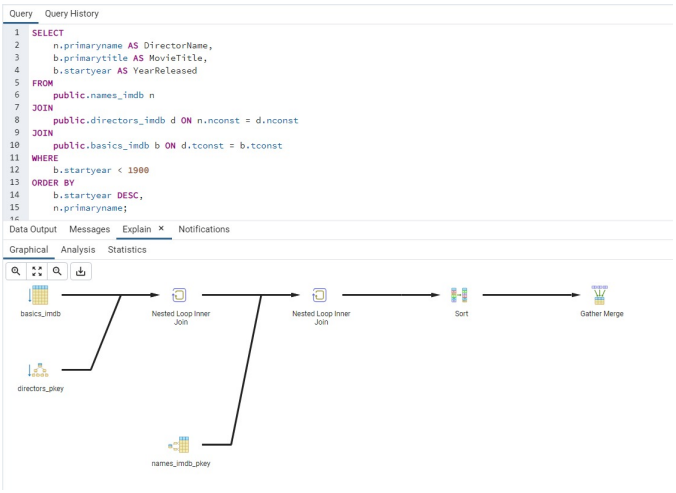


```
Query  Query History
1  SELECT
2      n.primaryname AS DirectorName,
3      b.primarytitle AS MovieTitle,
4      b.startyear AS YearReleased
5  FROM
6      public.names_imdb n
7  JOIN
8      public.directors_imdb d ON n.nconst = d.nconst
9  JOIN
10     public.basics_imdb b ON d.tconst = b.tconst
11 WHERE
12     b.startyear < 1900
13 ORDER BY
14     b.startyear DESC,
15     n.primaryname;
```

Data Output Messages Explain × Notifications

	directorname character varying (500)	movietitle character varying (600)	yearreleased integer
1	Afonso Segreto	Um Batalhão do Exército	1899
2	Afonso Segreto	Quadros Nacionais	1899
3	Afonso Segreto	Praça Tamarindo no Dia Treze de Maio	1899
4	Afonso Segreto	Uma Viagem de Núpcias Que Acaba Mal	1899
5	Afonso Segreto	Um Careca	1899
6	Afonso Segreto	Rua do Ouvidor	1899
7	Afonso Segreto	Algumas Localidades do Rio de Janeiro	1899
8	Afonso Segreto	Entrada de Uma Barca de Niterói	1899
9	Afonso Segreto	Festas em Homenagem ao Presidente da Argentina General Julio Roca	1899
10	Afonso Segreto	Dança de um Baiano	1899
11	Afonso Segreto	Embarque do Dr. Campos Sales Para Petrópolis	1899
12	Afonso Segreto	Enterro do Primeiro Tenente Pio Torelli	1899
13	Afonso Segreto	O Mágico dos Bonecos	1899
14	Afonso Segreto	Grandes Solenidades Comemorativas de Colônia Italiana no Rio	1899

Total rows: 1000 of 2326 Query complete 00:00:02.113

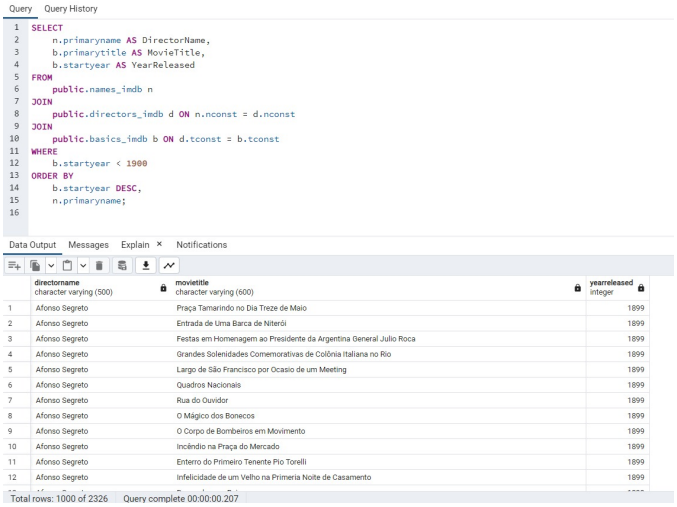


```
Query  Query History
1  SELECT
2      n.primaryname AS DirectorName,
3      b.primarytitle AS MovieTitle,
4      b.startyear AS YearReleased
5  FROM
6      public.names_imdb n
7  JOIN
8      public.directors_imdb d ON n.nconst = d.nconst
9  JOIN
10     public.basics_imdb b ON d.tconst = b.tconst
11 WHERE
12     b.startyear < 1900
13 ORDER BY
14     b.startyear DESC,
15     n.primaryname;
```

Data Output Messages Explain × Notifications

Graphical Analysis Statistics

- *startup cost*: 235652.52
- *total cost*: 235991.74
- *Execution time*: 2.113seconds
- *Scan*: Parallel Sequential Scan on basics_imdb

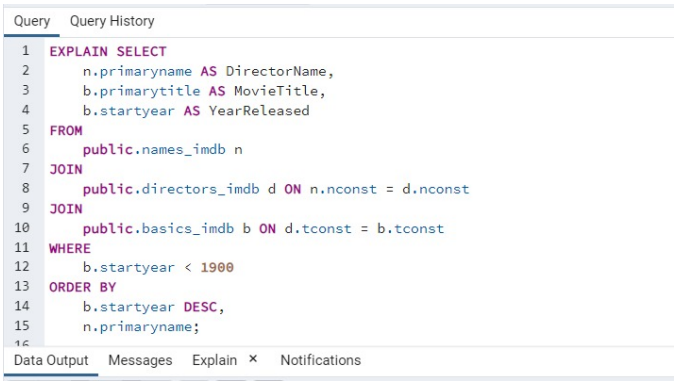


```
Query  Query History
1  SELECT
2      n.primaryname AS DirectorName,
3      b.primarytitle AS MovieTitle,
4      b.startyear AS YearReleased
5  FROM
6      public.names_imdb n
7  JOIN
8      public.directors_imdb d ON n.nconst = d.nconst
9  JOIN
10     public.basics_imdb b ON d.tconst = b.tconst
11 WHERE
12     b.startyear < 1900
13 ORDER BY
14     b.startyear DESC,
15     n.primaryname;
16
```

Data Output Messages Explain × Notifications

	directorname character varying (500)	movietitle character varying (600)	yearreleased integer
1	Afonso Segreto	Praça Tamarindo no Dia Treze de Maio	1899
2	Afonso Segreto	Entrada de Uma Barca de Niterói	1899
3	Afonso Segreto	Festas em Homenagem ao Presidente da Argentina General Julio Roca	1899
4	Afonso Segreto	Grandes Solenidades Comemorativas de Colônia Italiana no Rio	1899
5	Afonso Segreto	Largo de São Francisco por Ocasio de um Meeting	1899
6	Afonso Segreto	Quadros Nacionais	1899
7	Afonso Segreto	Rua do Ouvidor	1899
8	Afonso Segreto	O Mágico dos Bonecos	1899
9	Afonso Segreto	O Corpo de Bombeiros em Movimento	1899
10	Afonso Segreto	Incêndio na Praça do Mercado	1899
11	Afonso Segreto	Enterro do Primeiro Tenente Pio Torelli	1899
12	Afonso Segreto	Infelicidade de um Velho na Primeria Noite de Casamento	1899

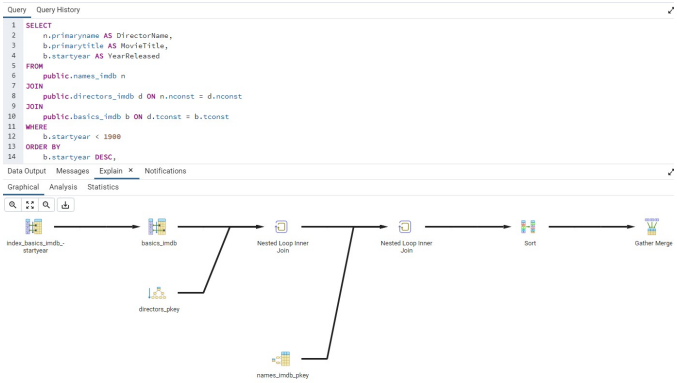
Total rows: 1000 of 2326 Query complete 00:00:00.207



```
Query  Query History
1  EXPLAIN SELECT
2      n.primaryname AS DirectorName,
3      b.primarytitle AS MovieTitle,
4      b.startyear AS YearReleased
5  FROM
6      public.names_imdb n
7  JOIN
8      public.directors_imdb d ON n.nconst = d.nconst
9  JOIN
10     public.basics_imdb b ON d.tconst = b.tconst
11 WHERE
12     b.startyear < 1900
13 ORDER BY
14     b.startyear DESC,
15     n.primaryname;
16
```

Data Output Messages Explain × Notifications

	QUERY PLAN text
1	Gather Merge (cost=235652.52..235992.74 rows=2916 width=38)
2	Workers Planned: 2
3	-> Sort (cost=234652.49..234656.14 rows=1458 width=38)
4	Sort Key: b.startyear DESC, n.primaryname
5	-> Nested Loop (cost=0.99..234575.88 rows=1458 width=38)
6	-> Nested Loop (cost=0.56..233780.46 rows=1458 width=34)
7	-> Parallel Seq Scan on basics_imdb b (cost=0.00..215640.39 rows=1871 width=34)
8	Filter: (startyear < 1900)
9	-> Index Only Scan using directors_pkey on directors_imdb d (cost=0.56..9.65 rows=5 width=...
10	Index Cond: (tconst = (b.tconst)::text)
11	-> Index Scan using names_imdb_pkey on names_imdb n (cost=0.43..0.55 rows=1 width=24)
12	Index Cond: ((nconst)::text = (d.nconst)::text)



```
Query  Query History
1  SELECT
2      n.primaryname AS DirectorName,
3      b.primarytitle AS MovieTitle,
4      b.startyear AS YearReleased
5  FROM
6      public.names_imdb n
7  JOIN
8      public.directors_imdb d ON n.nconst = d.nconst
9  JOIN
10     public.basics_imdb b ON d.tconst = b.tconst
11 WHERE
12     b.startyear < 1900
13 ORDER BY
14     b.startyear DESC,
```

Data Output Messages Explain × Notifications

Graphical Analysis Statistics

Query Query History

```
EXPLAIN SELECT
    n.primaryname AS DirectorName,
    b.primarytitle AS MovieTitle,
    b.startyear AS YearReleased
FROM
    public.names_imdb n
JOIN
    public.directors_imdb d ON n.nconst = d.nconst
JOIN
    public.basics_imdb b ON d.tconst = b.tconst
WHERE
    b.startyear < 1900
ORDER BY
    b.startyear DESC,
```

Data Output Messages Explain × Notifications

	QUERY PLAN text
1	Gather Merge (cost=35586.85..35927.08 rows=2916 width=38)
2	Workers Planned: 2
3	-> Sort (cost=34586.83..34590.47 rows=1458 width=38)
4	Sort Key: b.startyear DESC, n.primaryname
5	-> Nested Loop (cost=52.23..34510.21 rows=1458 width=38)
6	-> Nested Loop (cost=51.79..33714.80 rows=1458 width=34)
7	-> Parallel Bitmap Heap Scan on basics_imdb b (cost=51.23..15574.73 rows=1871 width=34)
8	Recheck Cond: (startyear < 1900)
9	-> Bitmap Index Scan on index_basics_imdb_startyear (cost=0.00..50.11 rows=4490 widt...
10	Index Cond: (startyear < 1900)
11	-> Index Only Scan using directors_pkey on directors_imdb d (cost=0.56..9.65 rows=5 width=...
12	Index Cond: (tconst = (b.tconst)::text)
13	-> Index Scan using names_imdb_pkey on names_imdb n (cost=0.43..0.55 rows=1 width=24)
14	Index Cond: ((nconst)::text = (d.nconst)::text)

Query Query History

```
    n.primaryname,
    n.birthyear,
    b.primarytitle,
    b.startyear,
    STRING_AGG(pp.primaryprofession, ', ') AS professions
FROM
    public.names_imdb n
JOIN
    public.knownfortitles_imdb kft ON n.nconst = kft.nconst
```

Data Output Messages Notifications

	QUERY PLAN text
1	GroupAggregate (cost=56.93..57.03 rows=4 width=74)
2	Group Key: n.birthyear, b.primarytitle, b.startyear
3	-> Sort (cost=56.93..56.94 rows=4 width=53)
4	Sort Key: n.birthyear DESC, b.primarytitle, b.startyear
5	-> Nested Loop (cost=1.99..56.89 rows=4 width=53)
6	Join Filter: ((n.nconst)::text = (pp.nconst)::text)
7	-> Nested Loop (cost=1.43..53.65 rows=4 width=62)
8	-> Nested Loop (cost=1.00..51.51 rows=4 width=48)
9	-> Index Scan using index_names_imdb_primaryname on names_imdb n (cost=0.56..24.65 rows=5 width=28)
10	Index Cond: ((primaryname)::text = 'Carla Politi'::text)
11	-> Index Only Scan using knownfortitles_imdb_pkey on knownfortitles_imdb kft (cost=0.43..5.32 rows=5 width...
12	Index Cond: (nconst = (n.nconst)::text)
13	-> Index Scan using basics_imdb_pkey on basics_imdb b (cost=0.43..0.53 rows=1 width=34)
14	Index Cond: ((tconst)::text = (kft.knownfortitle)::text)
15	-> Index Only Scan using primaryprofession_imdb_pkey on primaryprofession_imdb pp (cost=0.56..0.77 rows=3 width...
16	Index Cond: (nconst = (kft.nconst)::text)

- *startup cost*: 35586.85
- *total cost*: 35927.08
- *Execution time*: 0.207 seconds
- *Scan*: Parallel Bitmap Heap Scan on basics_imdb

- **SELECT n.primaryname, n.birthyear, b.primarytitle, b.startyear, STRING_AGG(pp.primaryprofession, ', ') AS professions FROM public.names_imdb n JOIN public.knownfortitles_imdb kft ON n.nconst = kft.nconst JOIN public.basics_imdb b ON kft.knownfortitle = b.tconst JOIN public.primaryprofession_imdb pp ON n.nconst = pp.nconst WHERE n.primaryname = 'Carla Politi' GROUP BY n.primaryname, n.birthyear, b.primarytitle, b.startyear ORDER BY n.birthyear DESC;** Before Indexing -

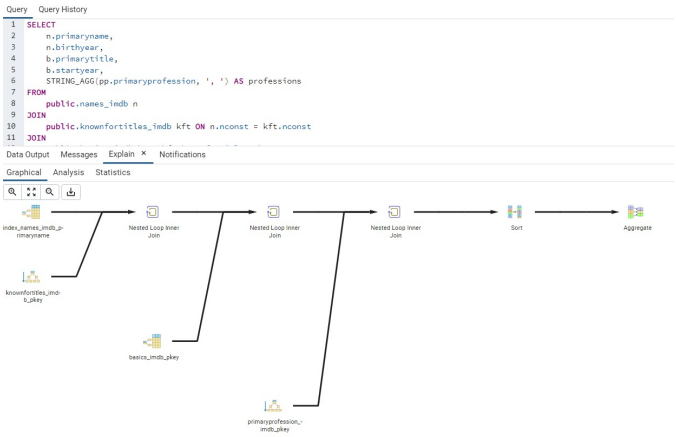
Query Query History

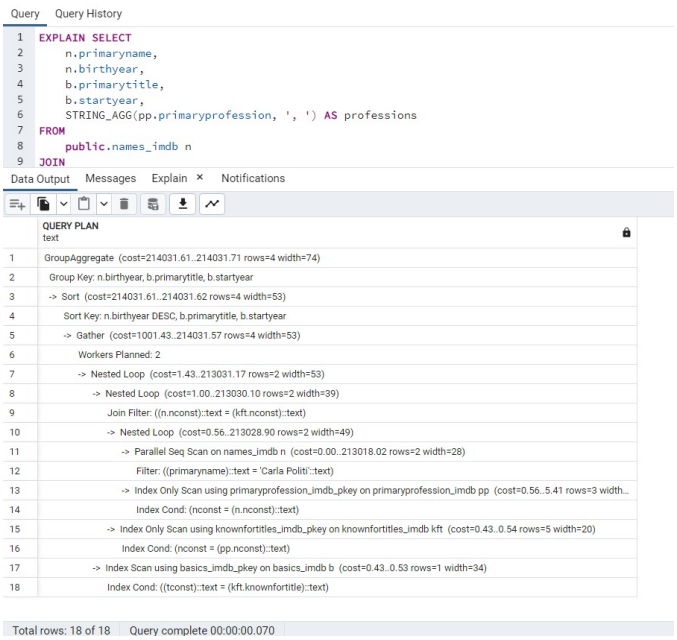
```
SELECT
    n.primaryname,
    n.birthyear,
    b.primarytitle,
    b.startyear,
    STRING_AGG(pp.primaryprofession, ', ') AS professions
FROM
    public.names_imdb n
JOIN
    public.knownfortitles_imdb kft ON n.nconst = kft.nconst
JOIN
    public.basics_imdb b ON kft.knownfortitle = b.tconst
JOIN
    public.primaryprofession_imdb pp ON n.nconst = pp.nconst
WHERE
    n.primaryname = 'Carla Politi'
GROUP BY
    n.primaryname, n.birthyear, b.primarytitle, b.startyear
ORDER BY
    n.birthyear DESC;
```

Data Output Messages Notifications

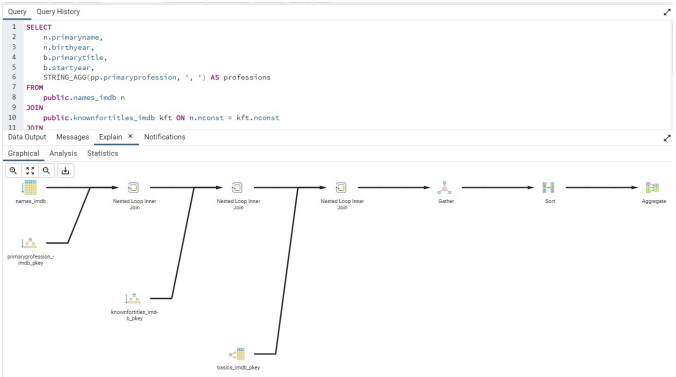
primaryname character varying (500)	birthyear integer	primarytitle character varying (600)	startyear integer	professions text

Total rows: 0 of 0 Query complete 00:00:00.057

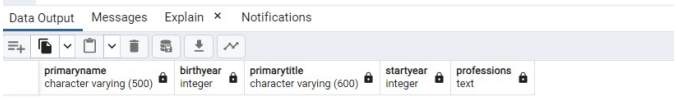




- *startup cost*: 214031.61
- *total cost*: 214031.71
- *Execution time*: 2.682 seconds
- *Scan*: Parallel Seq Scan on names_imdb n



```
SELECT
    n.primaryname,
    n.birthyear,
    b.primarytitle,
    b.startyear,
    STRING_AGG(pp.primaryprofession, ', ') AS professions
FROM
    public.names_imdb n
JOIN
    public.knownfortitles_imdb kft ON n.nconst = kft.nconst
JOIN
    public.basics_imdb b ON kft.knownfortitle = b.tconst
JOIN
    public.primaryprofession_imdb pp ON n.nconst = pp.nconst
WHERE
    n.primaryname = 'Carla Politi'
GROUP BY
    n.primaryname, n.birthyear, b.primarytitle, b.startyear
ORDER BY
    n.birthyear DESC;
```



- *startup cost*: 56.93
- *total cost*: 57.03
- *Execution time*: 0.057 seconds
- *Scan*: Index scan on names_imdb n

IX. Contribution

Team Member	Contribution
Vishnu Jampala	Task 1,2,4,5,6
Sai Murali	Task 2,3,6,7,8
Srija	Task 2,3,5,6,8

References

[1] IMDb Data Files https://datasets.imdbws.com/