

# IMDb Movie Database

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

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

3<sup>rd</sup> 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.

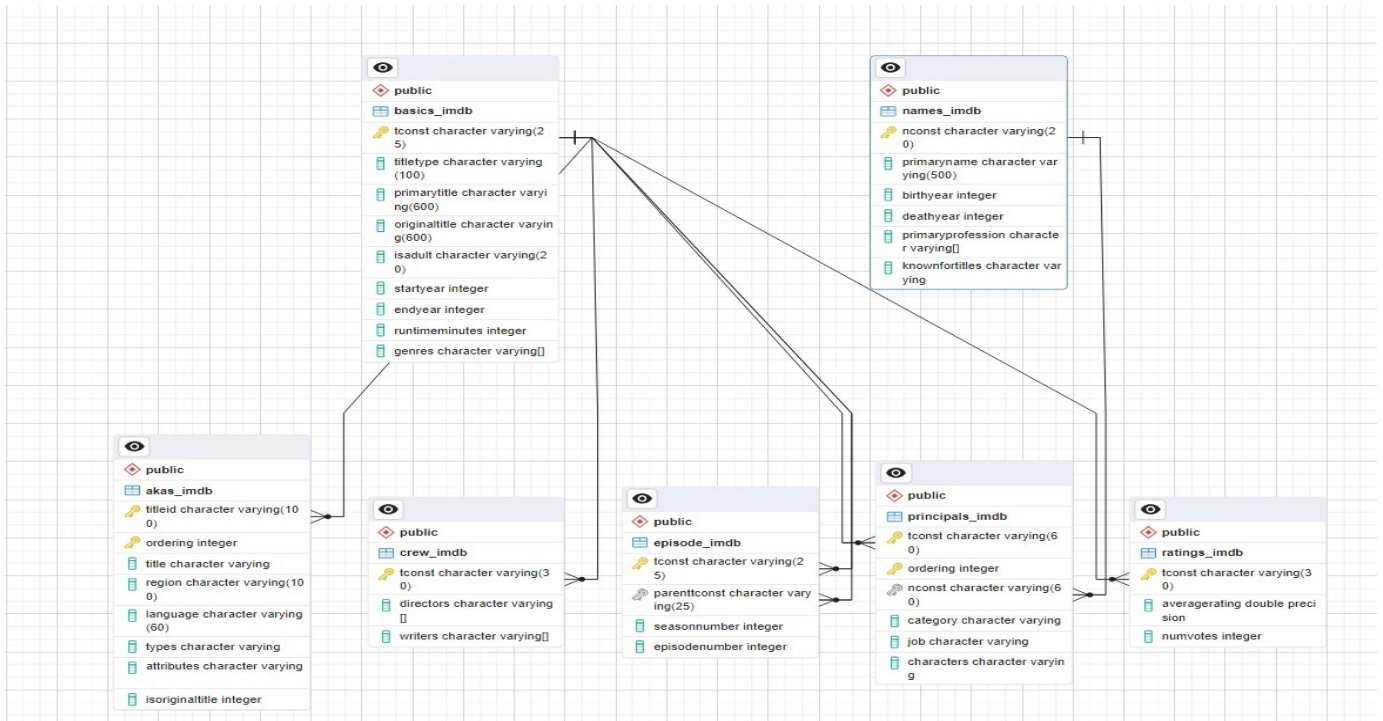


Fig. 1. E/R Diagram before BCNF

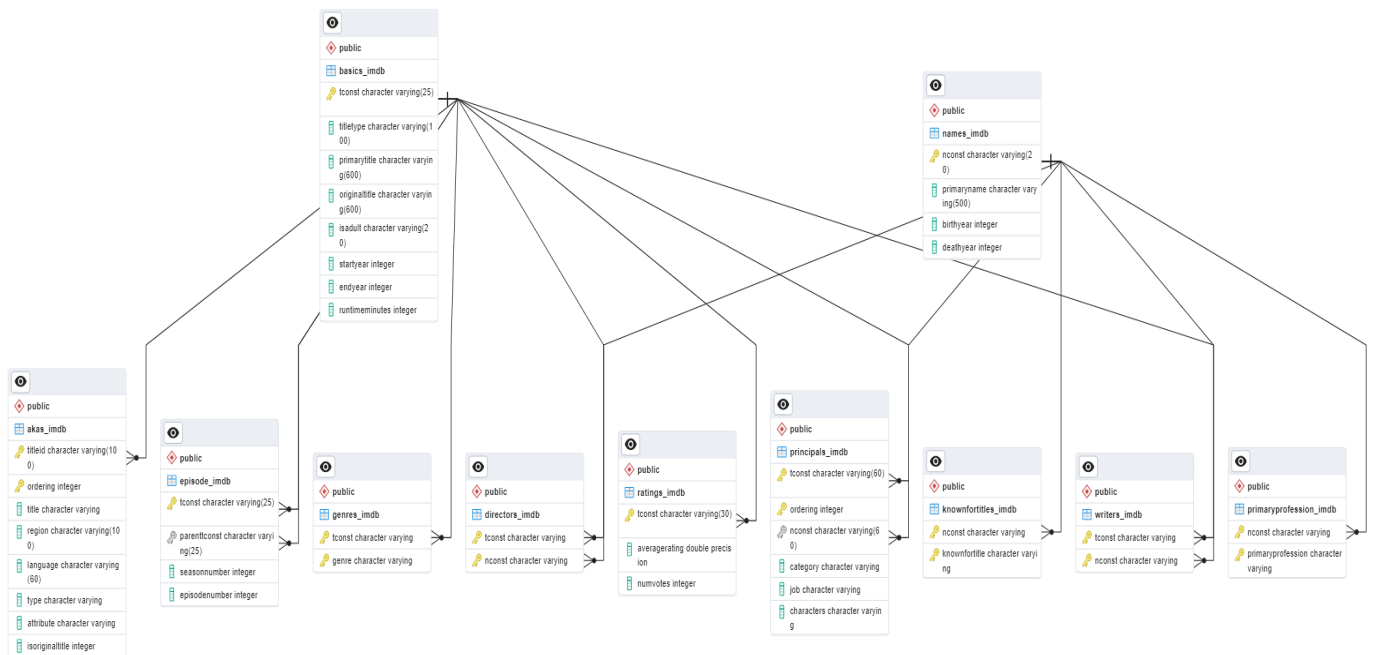


Fig. 2. E/R Diagram after BCNF

## VI. PROBLEMS WITH DATASET

### A. Loading Dataset

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

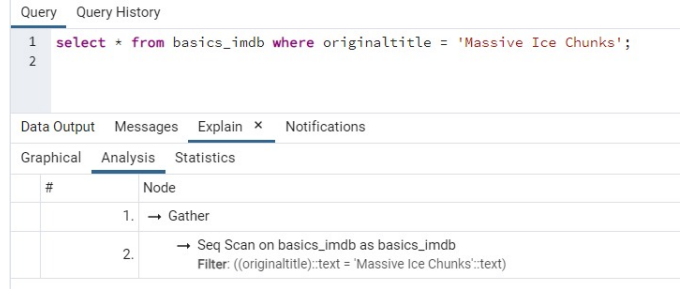
1) *Large dataset Size:* We had already identified and eliminated redundant data in our large dataset, which was time-consuming and complex due to multiple dependencies and functional relationships. Despite the challenges involved in dealing with such intricacies, we successfully addressed the issue 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)`

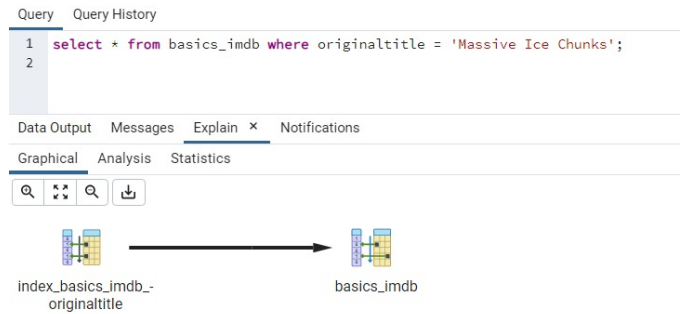
3) *Slower Data Retrieval:* For queries that require scanning the entire table, the execution time is significantly higher due to the large size of the dataset. However, indexing can benefit queries that exhibit prolonged execution times caused by extensive table-searching operations. By indexing the 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, runtimeinminutes) VALUES ('tt9999999', 'movie', 'New Movie', 'New Movie Original', '0', 2023, NULL, 120);

Query	Query History
1	INSERT INTO public.basics_imdb (tconst, titletype, primarytitle, originaltitle, isadult, startyear, endyear, runtimeinminutes)
2	VALUES ('tt9999999', 'movie', 'New Movie', 'New Movie Original', '0', 2023, NULL, 120);
3	
4	select * from basics_imdb where tconst = 'tt9999999';
5	

Data Output	Messages	Notifications
UPDATE 1		
Query returned successfully in 1 secs 66 msec.		

tconst	titletype	primarytitle	originaltitle	isadult	startyear	endyear	runtimeinminutes
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');

Query	Query History
1	INSERT INTO public.genres_imdb (tconst, genre)
2	VALUES ('tt9999999', 'Drama'), ('tt9999999', 'Thriller');
3	
4	select * from genres_imdb where tconst = 'tt9999999';
5	

Data Output	Messages	Notifications
UPDATE 2		
Query returned successfully in 1 secs 66 msec.		

tconst	genre
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';

Query	Query History
1	UPDATE public.basics_imdb SET primarytitle = 'Updated Movie Title' WHERE tconst = 'tt9999999';
2	

Data Output	Messages	Notifications
UPDATE 1		
Query returned successfully in 48 msec.		

Query	Query History
1	UPDATE public.basics_imdb SET primarytitle = 'Updated Movie Title' WHERE tconst = 'tt9999999';
2	
3	select * from basics_imdb where tconst = 'tt9999999';
4	

Data Output	Messages	Notifications
UPDATE 1		
Query returned successfully in 48 msec.		

tconst	titletype	primarytitle	originaltitle	isadult	startyear	endyear	runtimeinminutes
tt9999999	movie	Updated Movie Title	New Movie Original	0	2023	[null]	120

- 4) Update the run time of a movie in basics\_imdb:

**Query:** UPDATE public.basics\_imdb SET runtimeinminutes = 130 WHERE tconst = 'tt9999999';

Query	Query History
1	UPDATE public.basics_imdb SET runtimeinminutes = 130 WHERE tconst = 'tt9999999';
2	
3	

Data Output	Messages	Notifications
UPDATE 1		
Query returned successfully in 1 secs 66 msec.		

Query	Query History
1	UPDATE public.basics_imdb SET runtimeinminutes = 130 WHERE tconst = 'tt9999999';
2	
3	
4	select * from basics_imdb where tconst = 'tt9999999';
5	

Data Output	Messages	Notifications
UPDATE 1		
Query returned successfully in 1 secs 66 msec.		

tconst	titletype	primarytitle	originaltitle	isadult	startyear	endyear	runtimeinminutes
tt9999999	movie	Updated Movie Title	New Movie Original	0	2023	[null]	130

- 5) Delete a title from basics\_imdb

**Query:** DELETE FROM public.basics\_imdb WHERE tconst = 'tt9999999';

Query	Query History
1	DELETE FROM public.basics_imdb WHERE tconst = 'tt9999999';
2	
3	
4	
5	

Data Output	Messages	Notifications
DELETE 1		
ERROR: Key (tconst)=(tt9999999) is still referenced from table "genres_imdb".update or delete on table "basics_imdb" violates foreign key constraint "genres_imdb_tconst_fkey" on table "genres_imdb"		
ERROR: update or delete on table "basics_imdb" violates foreign key constraint "genres_imdb_tconst_fkey" on table "genres_imdb"		
SQL state: 23503		
Detail: Key (tconst)=(tt9999999) is still referenced from table "genres_imdb".		

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.genres_imdb WHERE tconst = 'tt9999999';
2	
3	
4	

Data Output	Messages	Notifications
DELETE 2		
Query returned successfully in 123 msec.		

Query	Query History
1	DELETE FROM public.basics_imdb WHERE tconst = 'tt9999999';
2	
3	
4	

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;

Query

Query History

1

2

3

4

5

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;

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

Query										Query History	
1										select * from basics_imdb where startyear=1894	
Data Output			Messages		Notifications						
<div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div></div>											
	isconst	startyear	endyear	runtime	minutes	seconds	milliseconds	microseconds	nanoseconds		
	character varying (20)	integer	integer	minutes	seconds	milliseconds	microseconds	nanoseconds	nanoseconds		
1	80000001	short	Carmenita	Carmenita	0	1894	[null]	1			
2	80000006	short	Chinese Opium Can	Chinese Opium Can	0	1894	[null]	1			
3	80000007	short	Corbett and Courtney Ref.	Corbett and Courtney Ref.	0	1894	[null]	1			
4	80000008	short	Edison Kinetoscope Rec.	Edison Kinetoscope Rec.	0	1894	[null]	1			
5	80000009	movie	Miss Jerry	Miss Jerry	0	1894	[null]	45			
6	80000015	short	Author d'une cabine	Author d'une cabine	0	1894	[null]	2			
7	10177707	short	Dickson Experimental So.	Dickson Experimental Sound.	0	1894	[null]	1			
8	10243409	short	Armand Dray	Armand Dray	0	1894	[null]	1			
9	10243524	short	Albino with Head	Albino with Head	0	1894	[null]	1			
10	10243537	short	A Bar Room Scene	A Bar Room Scene	0	1894	[null]	1			
11	10243572	short	Bertoldi (Mouth Support)	Bertoldi (Mouth Support)	0	1894	[null]	1			
12	10243573	short	Bertoldi (Talis Contortion)	Bertoldi (Talis Contortion)	0	1894	[null]	1			
13	10243585	short	Boxing	Boxing	0	1894	[null]	1			
14	10243598	short	Boxing Match	Boxing Match	0	1894	[null]	1			
15	10243623	short	The Cock Fight	The Cock Fight	0	1894	[null]	1			
16	10244054	short	Fred Ott Holding a Bird	Fred Ott Holding a Bird	0	1894	[null]	1			
17	10244071	short	Men on Parallel Bars	Men on Parallel Bars	0	1894	[null]	1			
18	10244086	short	Trained Bears	Trained Bears	0	1894	[null]	5			
19	10244179	short	Unsuccessful Somersault	Unsuccessful Somersault	0	1894	[null]	1			
20	10244185	short	Amiable Sun Dance	Amiable Sun Dance	0	1894	[null]	1			
21	10244282	short	Caccedo (with Pole)	Caccedo (with Pole)	0	1894	[null]	1			
22	10244283	short	Caccedo (with Spurs)	Caccedo (with Spurs)	0	1894	[null]	1			
23	10244289	short	Edison Electric Phonc	Edison Electric Phonc	0	1894	[null]	1			
24	10244306	short	Ruth Dennis	Ruth Dennis	0	1894	[null]	1			
Total rows: 100 of 100    Query complete 00:00:03.408											
										Ln 1, Col 4	

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;

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;



Query	Query History
1	SELECT genre, COUNT(tconst) AS title_count
2	FROM public.genres_imdb
3	GROUP BY genre;
4	

Data Output	Messages	Notifications
<div> <div>genre</div> <div>character varying</div> </div> <div> <div>title_count</div> <div>bigint</div> </div>		
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

Query	Query History
1	SELECT g.genre, b.primarytitle
2	FROM public.genres_imdb g
3	LEFT JOIN public.basics_imdb b ON g.tconst = b.tconst;
4	

Data Output	Messages	Notifications
<div> <div>genre</div> <div>character varying</div> </div> <div> <div>primarytitle</div> <div>character varying (500)</div> </div>		
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

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');

Query	Query History
1	SELECT n.primaryname
2	FROM public.names_imdb n
3	WHERE n.nconst IN (SELECT d.nconst FROM public.directors_imdb d WHERE d.tconst = 'tt9999999');
4	

Data Output	Messages	Notifications
<div> <div>primaryname</div> <div>character varying (500)</div> </div>		
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;

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

```
1 SELECT b.primarytitle, n.primaryname, b.startyear
2 FROM public.basics_imdb b
3 JOIN public.directors_imdb d ON b.tconst = d.tconst
4 JOIN public.names_imdb n ON d.nconst = n.nconst
5 ORDER BY b.startyear DESC LIMIT 100;
6
```

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 Radheyilal 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 Radheyilal 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

Query

Query History

```
1 SELECT b.primarytitle, AVG(r.averagerating) AS AverageRating
2 FROM public.basics_imdb b
3 JOIN public.ratings_imdb r ON b.tconst = r.tconst
4 JOIN public.genres_imdb g ON b.tconst = g.tconst
5 WHERE g.genre = 'Drama'
6 GROUP BY b.primarytitle
7 HAVING COUNT(b.tconst) > 1
8 ORDER BY AverageRating DESC;
9
```

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	Tendiya	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	FP 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

```
1 SELECT e.parenttconst, e.seasonnumber, COUNT(e.episodenumber) AS EpisodesCount
2 FROM public.episode_imdb e
3 WHERE e.parenttconst = 'tt0041038'
4 GROUP BY e.parenttconst, e.seasonnumber
5 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

#### 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;

#### 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;

Query Query History

```
1 SELECT b.primarytitle, AVG(r.averagerating) AS Averagerating, STRING_AGG(n1.primaryname, ',') AS Directors, STRING_AGG(n2.primaryname, ',') AS Writers
2 FROM public basics_imdb b
3 JOIN public ratings_imdb r ON b.tconst = r.tconst
4 LEFT JOIN public directors_imdb d ON b.tconst = d.tconst
5 LEFT JOIN public names_imdb n1 ON d.nconst = n1.nconst
6 LEFT JOIN public writers_imdb w ON b.tconst = w.tconst
7 LEFT JOIN public names_imdb n2 ON w.nconst = n2.nconst
8 GROUP BY b.primarytitle
9 HAVING AVG(r.averagerating) > 4.8
10 ORDER BY Averagerating DESC LIMIT 100;
```

Data Output Messages Explain × Notifications

primarytitle	averagerating	directors	writers
1 #Theatrical TV Edition Interview with Samuel Carl Cohen - Lacton	10	Samuel Carl Cohen, Leonardo Thimo	Leonardo Thimo, Leonardo Thimo
2 #Theatrical TV Edition Interview with Serkan Altun - The Librarian	10	Serkan Altun, Leonardo Thimo	Leonardo Thimo, Leonardo Thimo
3 #Theatrical TV Edition Interview with Barbara Lando - Obscure Meat Cake	10	Barbara Lando, Leonardo Thimo	Leonardo Thimo, Leonardo Thimo
4 #Theatrical TV Edition Interview with Nina Malkin - The meaning of life	10	Nina Malkin, Leonardo Thimo	Leonardo Thimo, Leonardo Thimo
5 #Theatrical TV Edition Interview with Sulaia Fernandes - THE MEMÓRIA DO PAI	10	Leonardo Thimo, Sulaia Fernandes	Leonardo Thimo, Leonardo Thimo
6 #Theatrical TV Edition Interview with Serkan Altun - Minority	10	Serkan Altun, Leonardo Thimo	Leonardo Thimo, Leonardo Thimo
7 #Theatrical TV Edition Interview with Maratya Yusuf - No Summer in the Area	10	Leonardo Thimo, Maratya Yusuf	Leonardo Thimo, Leonardo Thimo
8 #Theatrical TV Edition Interview with Mounad Hamid - When looking at you	10	Mounad Hamid, Leonardo Thimo	Leonardo Thimo, Leonardo Thimo
9 #Theatrical TV Edition Interview with Nathalie Dunsenman - Paper Talk	10	Leonardo Thimo	Leonardo Thimo
10 #Theatrical TV Edition Interview with Marco Cecchetti - Lobotom Islands	10	Leonardo Thimo, Marco Cecchetti	Leonardo Thimo, Leonardo Thimo
11 #Theatrical TV Edition Interview with Jin Peng - Side	10	Jin Peng, Leonardo Thimo	Leonardo Thimo, Leonardo Thimo
12 #Theatrical TV Edition Interview with Mateo Merchán Lopez - Ana	10	Mateo Merchán Lopez, Leonardo Thimo	Leonardo Thimo, Leonardo Thimo
13 #Theatrical TV Edition Interview with Ronald Bal - Catalysis	10	Leonardo Thimo, Ronald Bal	Leonardo Thimo, Leonardo Thimo
14 #Theatrical TV Edition Interview with Antonio Rodriguez - The King of the Hill	10	Antonio Rodriguez, Leonardo Thimo	Leonardo Thimo, Leonardo Thimo
15 #Theatrical TV Edition Interview with Jose Dirnberg - The King of the Hill	10	Leonardo Thimo, Jose Dirnberg	Leonardo Thimo, Leonardo Thimo
16 #Theatrical TV Edition Interview with Mayram Yadeget - Days	10	Leonardo Thimo, Mayram Yadeget	Leonardo Thimo, Leonardo Thimo

## 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 -

Query Query History

```
1 EXPLAIN select * from basics_imdb where primarytitle = 'Paparazzi';
```

Data Output Messages Explain × Notifications

QUERY PLAN

text

1	Gather (cost=1000.00..216652.69 rows=123 width=73)
2	Workers Planned: 2
3	-> Parallel Seq Scan on basics_imdb (cost=0.00..215640.39 rows=51 width=...
4	Filter: ((primarytitle)::text = 'Paparazzi')::text)

Query Query History

```
1 EXPLAIN select * from basics_imdb where primarytitle = 'Paparazzi';
```

Data Output Messages Explain × Notifications

QUERY PLAN

text

1	Bitmap Heap Scan on basics_imdb (cost=5.51..488.81 rows=123 width=...
2	Recheck Cond: ((primarytitle)::text = 'Paparazzi')::text)
3	-> Bitmap Index Scan on i (cost=0.00..5.48 rows=123 width=0)
4	Index Cond: ((primarytitle)::text = 'Paparazzi')::text)

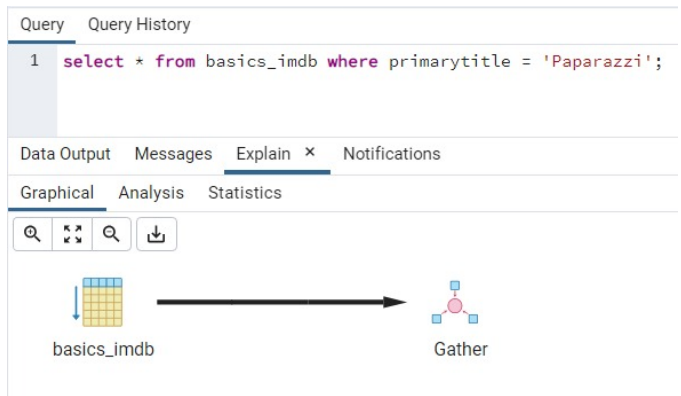
Query Query History

```
1 select * from basics_imdb where primarytitle = 'Paparazzi';
```

Data Output Messages Explain × Notifications

tconst	primarytitle	originaltitle	isadult	startyear	endyear	runtime	minutes
1	Paparazzi	Paparazzi	0	2009		101	8
2	Paparazzi	Paparazzi	0	2006		101	18
3	Paparazzi	Paparazzi	0	2012		101	44
4	Paparazzi	Paparazzi	0	2012		101	19
5	Paparazzi	Paparazzi	0	2013		101	101
6	Paparazzi	Paparazzi	0	2013		101	101
7	Paparazzi	Paparazzi	0	2013		101	101
8	Paparazzi	Paparazzi	0	2003		101	101
9	Paparazzi	Paparazzi	0	1997		101	101
10	Paparazzi	Paparazzi	0	2007		101	101
11	Paparazzi	Paparazzi	1	2018		101	140
12	Paparazzi	Paparazzi	0	1964		101	101
13	Paparazzi	Paparazzi	0	1964		101	101
14	Paparazzi	Paparazzi	0	1998		101	111
15	Paparazzi	Paparazzi	0	1997	1997	101	60
16	Paparazzi	Paparazzi	0	1998		101	100
17	Paparazzi	Paparazzi	0	2001		101	30
18	Paparazzi	Paparazzi	0	2015		101	101
19	Paparazzi	Paparazzi	0	2009		101	101
20	Paparazzi	Paparazzi	0	2006		101	101
21	Paparazzi	Paparazzi	1	2020		101	111
22	Paparazzi	Paparazzi	0	2003		101	101
23	Paparazzi	Paparazzi	0	2014		101	23
24	Paparazzi	Paparazzi	0	2009		101	101

Total rows: 64 of 64 Query complete 00:00:02.693



- **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 -

Query

Query History

1









EXPLAIN select \* from basics\_imdb where primarytitle = 'Paparazzi';

Data Output

Messages

Explain ×

Notifications



QUERY PLAN

text

1

Bitmap Heap Scan on basics\_imdb (cost=5.51..488.81 rows=123 width=...

2

Recheck Cond: ((primarytitle)::text = 'Paparazzi'::text)

3

-> Bitmap Index Scan on i (cost=0.00..5.48 rows=123 width=0)

4

Index Cond: ((primarytitle)::text = 'Paparazzi'::text)

Query Query History

1 select \* from basics\_imdb where primarytitle = 'Paparazzi';

Data Output Messages Explain × Notifications

	id	table	type	primarytitle	originaltitle	isadult	startyear	endyear	runtimeinminutes
1	10058448	short	character varying (25)	Paparazzi	Paparazzi	0	1964	[null]	18
2	1013314	movie	character varying (100)	Paparazzi	Paparazzi	0	1968	[null]	111
3	10173583	tvseries		Paparazzi	Paparazzi	0	1997	1997	60
4	10174105	movie		Paparazzi	Paparazzi	0	1998	[null]	100
5	10329905	tvseries		Paparazzi	Paparazzi	0	2001	[null]	30
6	10338525	movie		Paparazzi	Paparazzi	0	2004	[null]	84
7	10492313	movie		Paparazzi	Paparazzi	0	1993	[null]	90
8	10799552	tepisode		Paparazzi	Paparazzi	0	2004	[null]	[null]
9	10937261	tepisode		Paparazzi	Paparazzi	0	2007	[null]	43
10	11043302	tepisode		Paparazzi	Paparazzi	0	2007	[null]	[null]
11	110951316	tepisode		Paparazzi	Paparazzi	0	2015	[null]	[null]
12	111236282	tepisode		Paparazzi	Paparazzi	0	2019	[null]	[null]
13	11142307	tepisode		Paparazzi	Paparazzi	0	2009	[null]	[null]
14	11194775	tepisode		Paparazzi	Paparazzi	0	2006	[null]	[null]
15	111987378	video		Paparazzi	Paparazzi	1	2020	[null]	111
16	112749352	video		Paparazzi	Paparazzi	1	2014	[null]	109
17	112794026	tepisode		Paparazzi	Paparazzi	0	2016	[null]	[null]
18	112965386	tmovie		Paparazzi	Paparazzi	0	1998	[null]	[null]
19	11373961	tepisode		Paparazzi	Paparazzi	0	2003	[null]	[null]
20	114037126	short		Paparazzi	Paparazzi	0	2014	[null]	23
21	114278194	tepisode		Paparazzi	Paparazzi	0	2021	[null]	[null]
22	114488948	tvseries		Paparazzi	Paparazzi	0	2021	[null]	24
23	114487078	tepisode		Paparazzi	Paparazzi	0	2009	[null]	[null]
24	11446870	movie		Paparazzi	Paparazzi	0	1969	[null]	55

Total rows: 64 of 64    Query complete 00:00:00.204

Query

Query History

1

select \* from basics\_imdb where primarytitle = 'Paparazzi';

Data Output

Messages





Explain ×


Notifications

Graphical

Analysis

Statistics





```
graph LR; Index[index_basics_imdb_primarytitle] --> Table[basics_imdb]
```

index\_basics\_imdb\_  
primarytitle

basics\_imdb

Query	Query History
1	<code>select * from basics_imdb where primarytitle = 'Paparazzi';</code>
<div><div>Data Output</div><div>Messages</div><div>Explain <span>×</span></div><div>Notifications</div></div>	
<div><div>Graphical</div><div>Analysis</div><div>Statistics</div></div>	
#	Node
1.	→ Bitmap Heap Scan on basics_imdb as basics_imdb Recheck Cond: ((primarytitle)::text = 'Paparazzi'::text)
2.	→ Bitmap Index Scan using index_basics_imdb_primarytitle Index Cond: ((primarytitle)::text = 'Paparazzi'::text)

– *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;
16	
Data Output	Messages Explain × Notifications
1	directorname
1	character varying (500)
1	movieitle
1	character varying (600)
1	yearreleased
1	integer
1	1999
2	Afonso Segreto
2	Um Barulho do Exterior
3	Afonso Segreto
3	Quadros Nacionais
4	Afonso Segreto
4	Praça Tamarindo no Dia Treze de Maio
5	Afonso Segreto
5	Uma Viagem de Nípcias Que Acaba Mal
6	Afonso Segreto
6	Um Careca
7	Afonso Segreto
7	Rua do Ovidor
8	Afonso Segreto
8	Algumas Localidades do Rio de Janeiro
9	Afonso Segreto
9	Entrada de Uma Barca de Niterói
10	Afonso Segreto
10	Festas em Homenagem ao Presidente da Argentina General Julio Roca
11	Afonso Segreto
11	Dança de um Balano
12	Afonso Segreto
12	Embarque do Dr. Campos Sales Para Petrópolis
13	Afonso Segreto
13	Entero do Primeiro Tenente Pio Torelli
14	Afonso Segreto
14	O Mágico dos Bonecos
15	Afonso Segreto
15	Grandes Solenidades Comemorativas de Colônia Italiana no Rio
16	
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;
16	
Data Output	Messages Explain × Notifications
Graphical	Analysis Statistics
1	basics_imdb
1	director_name
1	movie_title
1	year_released
1	1999
2	Afonso Segreto
2	Um Barulho do Exterior
3	Afonso Segreto
3	Quadros Nacionais
4	Afonso Segreto
4	Praça Tamarindo no Dia Treze de Maio
5	Afonso Segreto
5	Uma Viagem de Nípcias Que Acaba Mal
6	Afonso Segreto
6	Um Careca
7	Afonso Segreto
7	Rua do Ovidor
8	Afonso Segreto
8	Algumas Localidades do Rio de Janeiro
9	Afonso Segreto
9	Entrada de Uma Barca de Niterói
10	Afonso Segreto
10	Festas em Homenagem ao Presidente da Argentina General Julio Roca
11	Afonso Segreto
11	Dança de um Balano
12	Afonso Segreto
12	Embarque do Dr. Campos Sales Para Petrópolis
13	Afonso Segreto
13	Entero do Primeiro Tenente Pio Torelli
14	Afonso Segreto
14	O Mágico dos Bonecos
15	Afonso Segreto
15	Grandes Solenidades Comemorativas de Colônia Italiana no Rio
16	
Total rows: 1000 of 2326	Query complete 00:00:02.113

- 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
1	directorname
1	character varying (500)
1	movieitle
1	character varying (600)
1	yearreleased
1	integer
1	1999
2	Afonso Segreto
2	Um Barulho do Exterior
3	Afonso Segreto
3	Quadros Nacionais
4	Afonso Segreto
4	Praça Tamarindo no Dia Treze de Maio
5	Afonso Segreto
5	Uma Viagem de Nípcias Que Acaba Mal
6	Afonso Segreto
6	Um Careca
7	Afonso Segreto
7	Rua do Ovidor
8	Afonso Segreto
8	Algumas Localidades do Rio de Janeiro
9	Afonso Segreto
9	Entrada de Uma Barca de Niterói
10	Afonso Segreto
10	Festas em Homenagem ao Presidente da Argentina General Julio Roca
11	Afonso Segreto
11	Dança de um Balano
12	Afonso Segreto
12	Embarque do Dr. Campos Sales Para Petrópolis
13	Afonso Segreto
13	Entero do Primeiro Tenente Pio Torelli
14	Afonso Segreto
14	O Mágico dos Bonecos
15	Afonso Segreto
15	Grandes Solenidades Comemorativas de Colônia Italiana no Rio
16	
Total rows: 1000 of 2326	Query complete 00:00:02.207

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
1	QUERY PLAN
1	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,
15	n.primaryname;
16	
Data Output	Messages Explain × Notifications
Graphical	Analysis Statistics
1	basics_imdb
1	director_name
1	movie_title
1	year_released
1	1999
2	Afonso Segreto
2	Um Barulho do Exterior
3	Afonso Segreto
3	Quadros Nacionais
4	Afonso Segreto
4	Praça Tamarindo no Dia Treze de Maio
5	Afonso Segreto
5	Uma Viagem de Nípcias Que Acaba Mal
6	Afonso Segreto
6	Um Careca
7	Afonso Segreto
7	Rua do Ovidor
8	Afonso Segreto
8	Algumas Localidades do Rio de Janeiro
9	Afonso Segreto
9	Entrada de Uma Barca de Niterói
10	Afonso Segreto
10	Festas em Homenagem ao Presidente da Argentina General Julio Roca
11	Afonso Segreto
11	Dança de um Balano
12	Afonso Segreto
12	Embarque do Dr. Campos Sales Para Petrópolis
13	Afonso Segreto
13	Entero do Primeiro Tenente Pio Torelli
14	Afonso Segreto
14	O Mágico dos Bonecos
15	Afonso Segreto
15	Grandes Solenidades Comemorativas de Colônia Italiana no Rio
16	
Total rows: 1000 of 2326	Query complete 00:00:02.207



Query	Query History
1	<b>EXPLAIN SELECT</b>
2	n.primaryname AS DirectorName,
3	b.primarytitle AS MovieTitle,
4	b.startyear AS YearReleased
5	<b>FROM</b>
6	public.names_imdb n
7	<b>JOIN</b>
8	public.directors_imdb d ON n.nconst = d.nconst
9	<b>JOIN</b>
10	public.basics_imdb b ON d.tconst = b.tconst
11	<b>WHERE</b>
12	b.startyear < 1900
13	<b>ORDER BY</b>
14	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 width=1)
10	Index Cond: (startyear < 1900)
11	-> Index Only Scan using directors_pkey on directors_imdb d (cost=0.56..9.65 rows=5 width=1)
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)

- 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
2	n.primaryname,
3	n.birthyear,
4	b.primarytitle,
5	b.startyear,
6	STRING_AGG(pp.primaryprofession, ', ') AS professions
7	<b>FROM</b>
8	public.names_imdb n
9	<b>JOIN</b>
10	public.knownfortitles_imdb kft ON n.nconst = kft.nconst
11	<b>JOIN</b>
12	public.basics_imdb b ON kft.knownfortitle = b.tconst
13	<b>JOIN</b>
14	public.primaryprofession_imdb pp ON n.nconst = pp.nconst
15	<b>WHERE</b>
16	n.primaryname = 'Carla Politi'
17	<b>GROUP BY</b>
18	n.primaryname, n.birthyear, b.primarytitle, b.startyear
19	<b>ORDER BY</b>
20	n.birthyear DESC;
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=1)
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=1)
16	Index Cond: (nconst = (kft.nconst)::text)

Query

Query History

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

```
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

birthyear

primarytitle

startyear

professions

character varying (500)

integer

character varying (600)

integer

text

Total rows: 0 of 0

Query complete 00:00:00.057

Query Query History

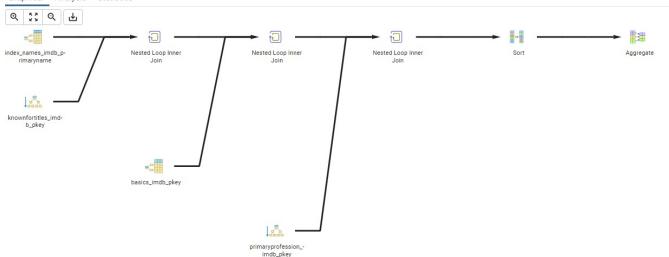
```

1 SELECT
2   n.primaryname,
3   n.birthyear,
4   b.primarytitle,
5   b.startyear,
6   STRING_AGG(pp.primaryprofession, ', ') AS professions
7 FROM
8   public.names_imdb n
9 JOIN
10  public.knownfortitles_imdb kft ON n.nconst = kft.nconst
11 JOIN
12  public.basics_imdb b ON kft.knownfortitle = b.tconst
13 JOIN
14  public.primaryprofession_imdb pp ON n.nconst = pp.nconst
15 WHERE
16   n.primaryname = 'Carla Politi'
17 GROUP BY
18   n.primaryname, n.birthyear, b.primarytitle, b.startyear
19 ORDER BY
20   n.birthyear DESC;

```

Data Output Messages Explain × Notifications

Graphical Analysis Statistics



- startup cost: 214031.61
- total cost: 214031.71
- Execution time: 2.682 seconds
- Scan: Parallel Seq Scan on names\_imdb n

Query Query History

```

1 EXPLAIN SELECT
2   n.primaryname,
3   n.birthyear,
4   b.primarytitle,
5   b.startyear,
6   STRING_AGG(pp.primaryprofession, ', ') AS professions
7 FROM
8   public.names_imdb n
9 JOIN
10  public.knownfortitles_imdb kft ON n.nconst = kft.nconst
11 JOIN
12  public.basics_imdb b ON kft.knownfortitle = b.tconst
13 JOIN
14  public.primaryprofession_imdb pp ON n.nconst = pp.nconst
15 WHERE
16   n.primaryname = 'Carla Politi'
17 GROUP BY
18   n.primaryname, n.birthyear, b.primarytitle, b.startyear
19 ORDER BY
20   n.birthyear DESC;

```

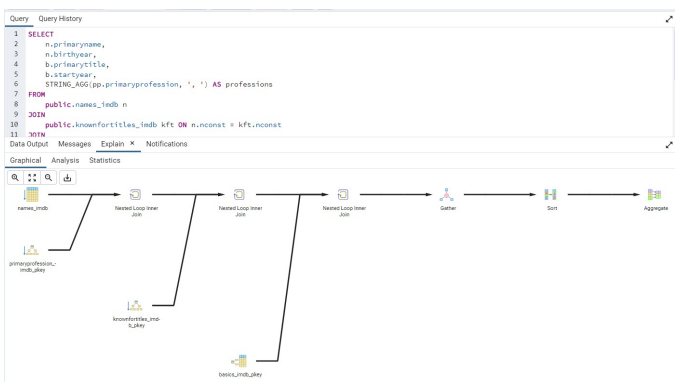
Data Output Messages Explain × Notifications

QUERY PLAN

text

- GroupAggregate (cost=214031.61..214031.71 rows=4 width=74)
- Group Key: n.birthyear, b.primarytitle, b.startyear
- > Sort (cost=214031.61..214031.62 rows=4 width=53)
- Sort Key: n.birthyear DESC, b.primarytitle, b.startyear
- > Gather (cost=1001.43..214031.57 rows=4 width=53)
- Workers Planned: 2
- > Nested Loop (cost=1.43..213031.17 rows=2 width=53)
- > Nested Loop (cost=1.00..213030.10 rows=2 width=99)
- Join Filter: ((n.nconst)::text = (kft.nconst)::text)
- > Nested Loop (cost=0.56..213028.90 rows=2 width=49)
- > Parallel Seq Scan on names\_imdb n (cost=0.00..213018.02 rows=2 width=28)
- Filter: ((primaryname)::text = 'Carla Politi')::text)
- > Index Only Scan using primaryprofession\_imdb\_pkey on primaryprofession\_imdb pp (cost=0.56..5.41 rows=3 width=...
- Index Cond: (nconst = (n.nconst)::text)
- > Index Only Scan using knownfortitles\_imdb\_pkey on knownfortitles\_imdb kft (cost=0.43..0.54 rows=5 width=20)
- Index Cond: (nconst = (pp.nconst)::text)
- > Index Scan using basics\_imdb\_pkey on basics\_imdb b (cost=0.43..0.53 rows=1 width=34)
- Index Cond: ((tconst)::text = (kft.knownfortitle)::text)

Total rows: 18 of 18 Query complete 00:00:00.070



- 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/>

Query Query History

```

1 SELECT
2   n.primaryname,
3   n.birthyear,
4   b.primarytitle,
5   b.startyear,
6   STRING_AGG(pp.primaryprofession, ', ') AS professions
7 FROM
8   public.names_imdb n
9 JOIN
10  public.knownfortitles_imdb kft ON n.nconst = kft.nconst
11 JOIN
12  public.basics_imdb b ON kft.knownfortitle = b.tconst
13 JOIN
14  public.primaryprofession_imdb pp ON n.nconst = pp.nconst
15 WHERE
16   n.primaryname = 'Carla Politi'
17 GROUP BY
18   n.primaryname, n.birthyear, b.primarytitle, b.startyear
19 ORDER BY
20   n.birthyear DESC;

```

Data Output Messages Explain × Notifications

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

Total rows: 0 of 0 Query complete 00:00:02.682