

# CSE – 560, Data Model and Query Language

## IMDB Dataset

### Milestone: 2

#### Project Details:

#### Team Crew - X

Name: Mahalakshmi Chintala  
UBIT: mchintal  
UBID: 50442308

Name: Harinee Purushothaman  
UBIT: harineep  
UBID: 50442192

Name: Anudeep Balagam  
UBIT: abalagam  
UBID: 50442091

#### I. INTRODUCTION

Not only does a popular movie amuse viewers, but it also makes huge profits for the film industry. A variety of elements, including talented directors and seasoned actors, are important for making good movies. Famous actors and filmmakers can, however, always bring in the anticipated box office revenue but cannot ensure a high imdb rating. So, we use this imdb rating system to tackle this problem.

#### II. PROBLEM STATEMENT

The IMDB dataset consists of the details of several movies, tv series and helps a user decide whether to watch the show or movie based on their ratings.

##### A. Usage of Database over Excel

- The use of spreadsheets for numeric and text values is effective in relatively small volumes. Also capable of incorporating other types of information, such as images and documents, databases can handle numeric and text values expertly.
- Since the IMDb Databases consists of more than a million records, it can also accommodate high volume and large file size data downloads. Generally, this includes those from data loggers, GPS devices, cameras, drones, and other collection devices.
- However, in the given scenario, it is easier to deal with data having nulls with a dataset over an excel sheet.

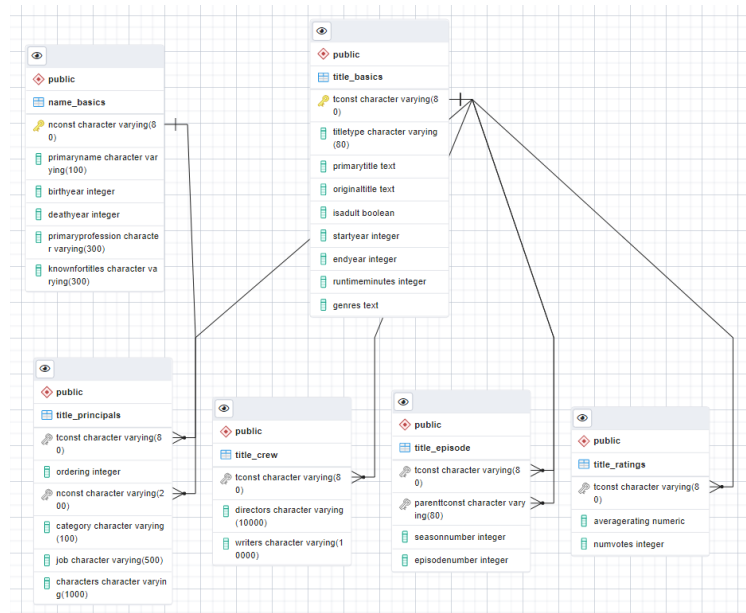
#### III. TARGET USER

General public or any enthusiast to find out information and ratings of different movies or series, short films, documentaries deciding to view it.

Also, another end user for imdb database system are users who want to rate or review movies or contribute data must sign up for an IMDb account.

The import and upkeep of the database platform fall under the purview of administrators.

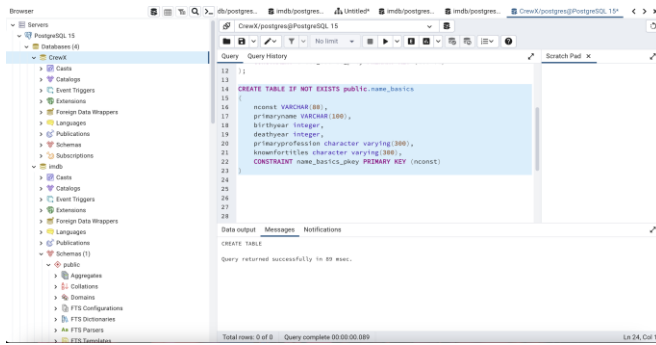
#### IV. ER DIAGRAM



##### A. Relationship between relations and attributes:

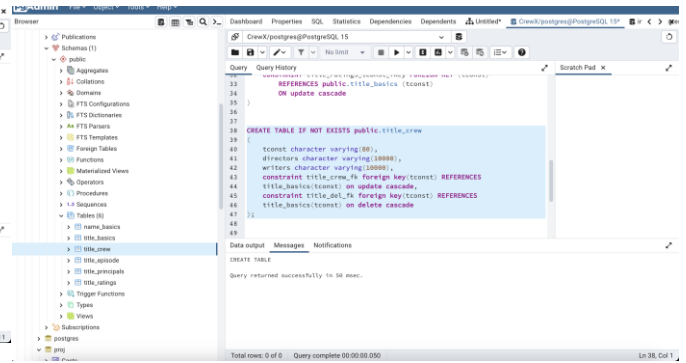
The **name\_basics** table consists of the below attributes:

- **nconst** (varchar) – Primary key, alphanumeric identifier of name of the artist.
- **primaryName** (varchar)– Most commonly credited name of the individual.
- **birthyear** (int) – Birth year of an individual artist.
- **deathyear** (int)– Death year of an individual, if applicable, else “\N”.
- **primaryProfession** (text) – This attribute is an array of text consists of the top-3 professions of the person.
- **knownForTitles** (text) – Titles the person is known for.



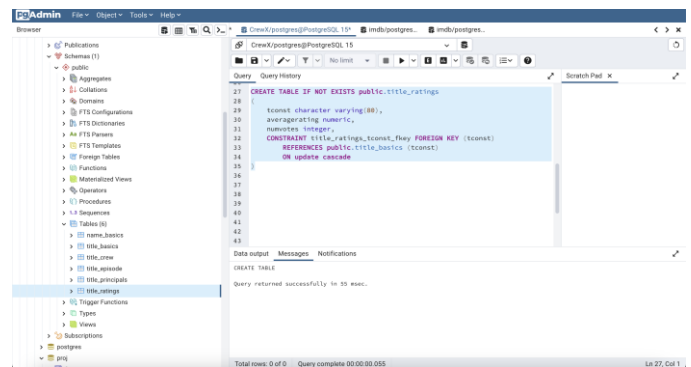
The **title\_basics** relation consists of the below:

- tconst (varchar) – Primary key of the table, which is an alphanumeric unique identifier of the title, equivalent to titleID.
- titletype (varchar) – format of the title
- primarytitle (varchar) – the title used by the filmmakers for promotions
- originaltitle (varchar) – Original title of the movie/show
- isadult (Boolean) – true i.e. 1 for adult movies and false i.e., 0 for the rest
- startyear (integer) – release year of the movie/ year of start of the series
- endyear (integer) – end year of tv series and ‘\N’ for all other types
- runtime (integer) – screentime of the title in minutes
- genres (text) – consists of up to 3 genres associated with the title



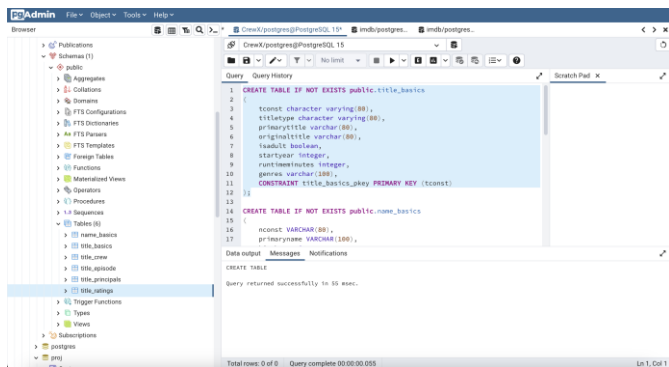
The **title\_ratings**: Listed below are the IMDb ratings and votes for the following titles.

- tconst (varchar) – title identifiers being references from tconst of title\_basics
- averageRating (numeric)– average of the user ratings
- numVotes (integer) - number of votes received by the title



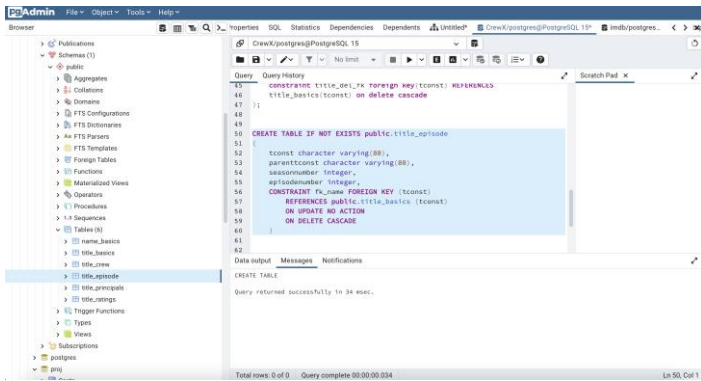
The **title\_episode** relation consists of the TV Series’ episodes information. Fields include:

- tconst (varchar) – episode identifiers being referenced from the tconst of the title\_basics
- parenttconst (varchar) –parent TV Series’ ID referred from tconst of title\_basics as foreign key
- seasonnumber (integer) – season number of the episode
- episodenum (integer) – episode number of the TV series



The **title\_crew** holds the director and writer data for all the titles in IMDb. Fields include:

- tconst (varchar) – title identifier which is a foreign key being referenced from tconst of title\_basics
- directors (varchar) - director(s) of the given title
- writers (varchar) – writer(s) of the given title



Similarly, if nconst of name\_basics is deleted, the records from title\_principals will automatically get deleted.

## V. BCNF

For a relation to be in BCNF, all of its functional dependents must be non-trivial, and the functional dependency's left side must be the relation's super key.

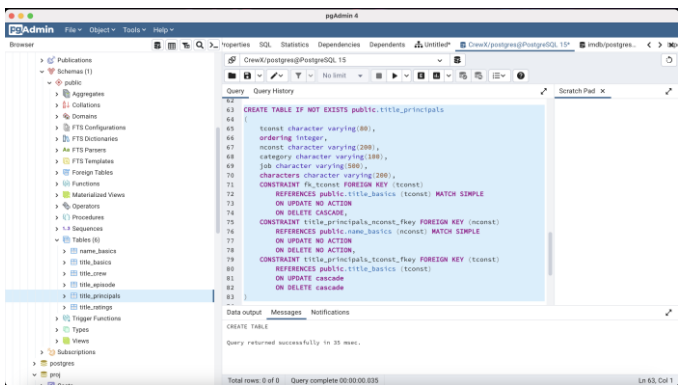
The initial IMDb schema is in BCNF already due to the following functional dependencies:

- Title.basics: Tconst-> title\_type, originaltitle, is\_adult, start\_year, runtime\_minutes, genre
- Names: Nconst-> primaryName, birth\_year, death\_year, primary\_profession
- Principals: Tconst, nconst -> ordering, category, job, characters
- Ratings: tconst-> average\_ratings, numvotes
- Crew: tconst-> directors, writers
- Episode: tconst-> parentTconst, seasonNumber, episodeNumber

There are additional possible functional dependencies but as they violate BCNF, we have considered only the above.

The **title\_principals** table has information about the titles' cast and/or crew, with the following attributes:

- tconst (varchar) – title identifiers referenced from tconst of the title\_basics relation
- ordering (integer) – a number to uniquely identify rows for a given titleId
- nconst (varchar) – foreign key of name identifier being referenced from the name\_basics table
- category (varchar) - the category of job that artist



## VI. PROBLEMS FACED WHILE USING THE LARGE DATASET:

- Recognizing nulls
- Size of tuples and attributes being larger
- Excess execution time for queries with more than one retrieving detail and from multiple tables

### A. Solutions:

- Altering query attributes expanding their size
- Indexing

Given that Postgres already produces an index for primary keys by default, it can be shown that when several attributes from the query are returned and indexed individually, the execution time is drastically reduced.

The usage of explain analyze helps understanding the query plan, process executed in running a query. EXPLAIN only supports SELECT, INSERT, UPDATE, DELETE, EXECUTE (of a prepared statement), CREATE TABLE... AS, and DECLARE. It does not handle other types of statements (of a cursor).

### B. Constraints:

The following attributes have the primary key constraint: nconst and tconst

The tconst and nconst attributes in the tables title\_crew, title\_ratings, title\_episode, title\_principals has been referenced from title\_basics and name\_basics as the foreign key.

The attributes primarytitle and isadult from the schema title\_basics is set to NOT NULL as a constraint.

### C. Delete Cascade:

The cascade delete feature of a foreign key ensures that when a record in the parent table is deleted, the corresponding records in the child table will also be deleted. SQL Server calls this a cascade delete.

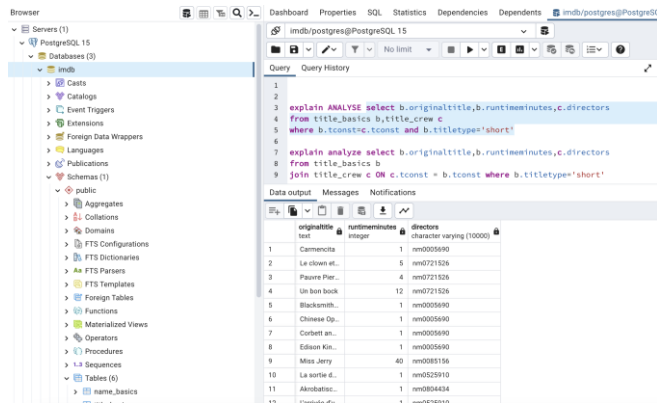
Hence, we have applied the delete cascade constraint over foreign keys present in title\_crew, title\_ratings, title\_episode, title\_principals.

On deletion of tconst tuple from the title\_basics table, tuples from the rest of the tables will also get deleted.

## VII. QUERIES

### A. SELECT Query 1

A query, to fetch the titles, their screen-time and directors for the title type is short.



The screenshot shows the PostgreSQL interface with a query executed. The query is:

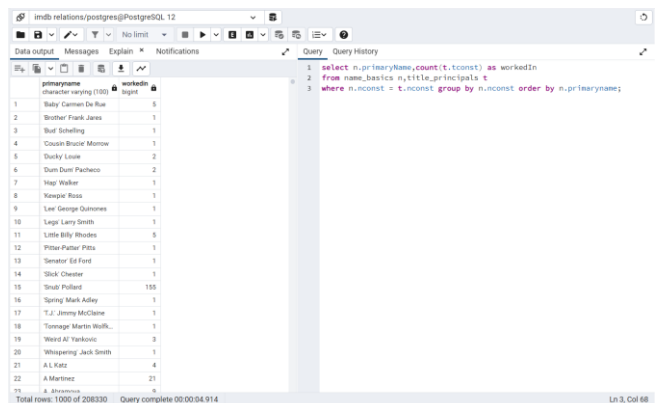
```
1 explain ANALYSE select b.originaltitle,b.runtime,b.directors
2 from title_basics b,title_ratings r
3 where b.tconst=r.tconst and b.titletype='short'
4
5 explain analyze select b.originaltitle,b.runtime,b.directors
6 from title_basics b
7 join title_crew c ON c.tconst = b.tconst where b.titletype='short'
```

The results are shown in a table with columns: originaltitle, runtime, and directors. The data is as follows:

originaltitle	runtime	directors
1 Camerita	1	nm0005690
2 Le clown et...	5	nm0721526
3 Paure Pier...	4	nm0721526
4 Un bon book	12	nm0721526
5 Blacksmith...	1	nm0005690
6 Chinese Op...	1	nm0005690
7 Corbett an...	1	nm0005690
8 Edison Kin...	1	nm0005690
9 Miss Jerry	40	nm0081156
10 La sortie d...	1	nm0525910
11 Akrobatie...	1	nm0804434

### B. SELECT Query 2

Finding name of people who have played more than one part in films along with their count



The screenshot shows the PostgreSQL interface with a query executed. The query is:

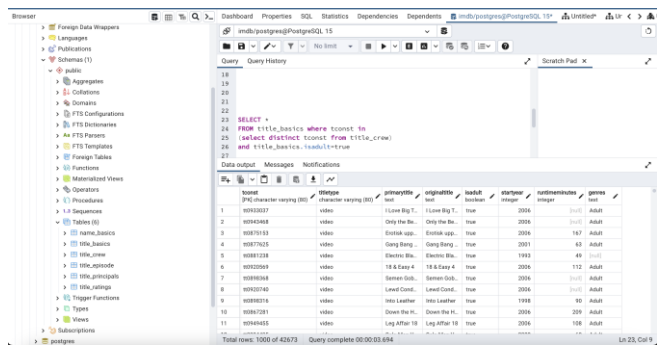
```
1 select n.primaryName,count(t.tconst) as workedIn
2 from name_basics n,title_principals t
3 where n.tconst = t.tconst group by n.primaryName;
```

The results are shown in a table with columns: primaryName and workedIn. The data is as follows:

primaryName	workedIn
1 Betty Carmen De Rue	5
2 Brother Frank Jones	1
3 Bud Schelling	1
4 Cousin Bruce Morrow	1
5 Ducky Louie	2
6 Sam Durr Pacheco	2
7 Meg Miller	1
8 Newbie Ross	1
9 Lee George Quinones	1
10 Legs Larry Smith	1
11 Little Billy Rhodes	5
12 Peter Foster Pitts	1
13 Senator Ed Ford	1
14 Slick Chester	1
15 Strub Pullard	105
16 Spring Mark Adley	1
17 T.J. Jimmy McClane	1
18 Tommye Marvin Hault	1
19 Ward Al Yankovic	3
20 Whispering Jack Smith	1
21 A.L.Katz	4
22 A Martinez	21

### C. SELECT Query 3

A query to find all the titles that are adult rated.



The screenshot shows the PostgreSQL interface with a query executed. The query is:

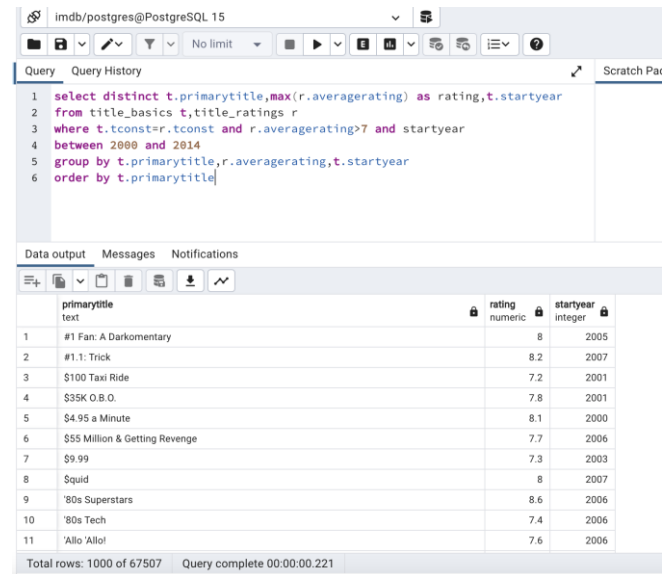
```
1 FROM title_basics where tconst in
2 (select distinct tconst from title_crew)
3 and title_basics.rating='R'
```

The results are shown in a table with columns: tconst, originaltitle, rating, and genre. The data is as follows:

tconst	originaltitle	rating	genre
1 00000001	1	R	Adult
2 00000002	2	R	Adult
3 00000003	3	R	Adult
4 00000004	4	R	Adult
5 00000005	5	R	Adult
6 00000006	6	R	Adult
7 00000007	7	R	Adult
8 00000008	8	R	Adult
9 00000009	9	R	Adult
10 00000010	10	R	Adult
11 00000011	11	R	Adult

### D. SELECT Query 4

Query to find movie titles that have an average rating of 7 and started in between the years 2000 to 2014.



The screenshot shows the PostgreSQL interface with a query executed. The query is:

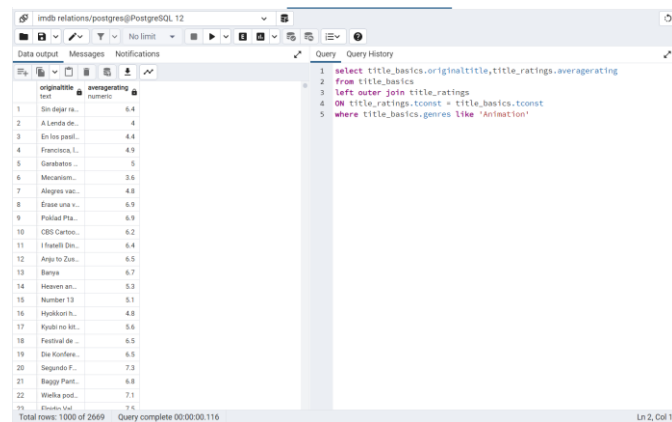
```
1 select distinct t.primarytitle,max(r.averagerating) as rating,t.startyear
2 from title_basics t,title_ratings r
3 where t.tconst=r.tconst and r.averagerating>7 and startyear
4 between 2000 and 2014
5 group by t.primarytitle,r.averagerating,t.startyear
6 order by t.primarytitle
```

The results are shown in a table with columns: primarytitle, rating, and startyear. The data is as follows:

primarytitle	rating	startyear
1 #1 Fan: A Darkomtory	8	2005
2 #1.1: Trick	8.2	2007
3 \$100 Taxi Ride	7.2	2001
4 \$35K O.B.O.	7.8	2001
5 \$4.95 a Minute	8.1	2000
6 \$55 Million & Getting Revenge	7.7	2006
7 \$9.99	7.3	2003
8 Squid	8	2007
9 '80s Superstars	8.6	2006
10 '80s Tech	7.4	2006
11 'Allo 'Allo!	7.6	2006

### E. SELECT Query 5

Query to find names of every animated movie along with their rating.



The screenshot shows the PostgreSQL interface with a query executed. The query is:

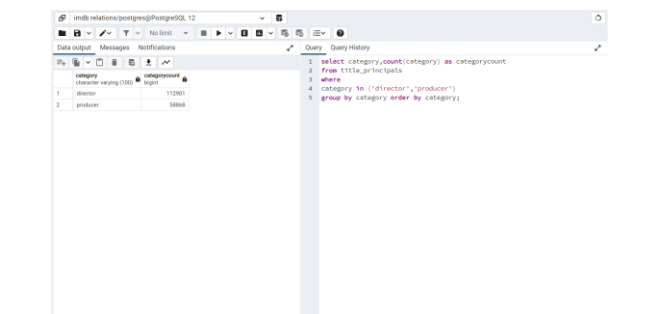
```
1 select title_basics.originaltitle,title_ratings.averagerating
2 from title_basics
3 left outer join title_ratings
4 on title_ratings.tconst = title_basics.tconst
5 where title_basics.genres like 'Animation'
```

The results are shown in a table with columns: originaltitle and averagerating. The data is as follows:

originaltitle	averagerating
1 Sin dejar ra...	6.4
2 A Lenda de...	4
3 En los past...	4.4
4 Rapaces L...	4.9
5 Garabatos ...	5
6 Mecanismo...	3.6
7 Aligned vac...	4.8
8 Erase una v...	6.9
9 Polkad Pla...	6.9
10 CBS Cartoo...	6.2
11 I Heart B...	6.4
12 Any to Zin...	6.5
13 Bambi	6.7
14 Heaven an...	5.3
15 Number 13	5.1
16 Hyokkori h...	4.8
17 Ryudo no B...	5.6
18 Festival de...	6.5
19 Die Koenige...	6.5
20 Segundo F...	7.3
21 Bagge Part...	6.8
22 Welka pod...	7.1
23 Chino no...	7.5

### F. SELECT Query 6

Query to find number of titles done by the director being same as writer.



The screenshot shows the PostgreSQL interface with a query executed. The query is:

```
1 select category,count(category) as categorycount
2 from title_principals
3 where
4 category in ('director','producer')
5 group by category order by category;
```

The results are shown in a table with columns: category and categorycount. The data is as follows:

category	categorycount
1 director	112001
2 producer	8888

## G. INSERT Query 7

Query to insert values into **name\_basics** table.

The screenshot shows a PostgreSQL Query Editor with the following SQL query:

```
1 insert into name_basics
2 values('nm5874513','Shae Krimson',1976,2002,'actor,producer',null)
3
4
5 select * from name_basics where primaryname='Shae Krimson';
```

The result shows a single row in the **name\_basics** table:

id	name_basics
1	nm5874513   Shae Krimson   1976   2002   actor,producer   [null]

Total rows: 1 of 1 | Query complete 00:00:00.103

## H. Update Query 8

Update the parentconst value to 'tt0989126' where seasonnumber = 1 and episodenumbr = 17 from **title\_episode**.

Before updating:

The screenshot shows a PostgreSQL Query Editor with the following SQL query:

```
1 select * from title_episode
2
```

The result shows a table with 11 rows and 4 columns: **id**, **title\_episode**, **parentconst**, and **seasonnumber**.

id	title_episode	parentconst	seasonnumber
1	tt0041951	tt0041038	1
2	tt0042816	tt0989125	1
3	tt0042819	tt0989125	1
4	tt0043426	tt0040051	3
5	tt0043631	tt0989125	2
6	tt0043693	tt0989125	2
7	tt0043710	tt0989125	3
8	tt0044093	tt0939862	1
9	tt0044668	tt0044243	2
10	tt0044901	tt0989125	3
11	tt0045519	tt0989125	4

Update:

The screenshot shows a PostgreSQL Query Editor with the following SQL query:

```
1 UPDATE title_episode set parentconst='tt0989126'
2 where seasonnumber = 1 and episodenumbr = 17
3
```

The result shows a message: "UPDATE 2/175. Query returned successfully in 694 msec."

After updating:

The screenshot shows a PostgreSQL Query Editor with the following SQL query:

```
1 select * from title_episode where seasonnumber=1 and
2 episodenumbr=17
```

The result shows a single row in the **title\_episode** table:

id	title_episode	parentconst	seasonnumber
1	tt0044759	tt0989126	1

## I. Delete Query 9

Deleting titles with no directors or writers within the title ids between 0134 to 0200.

The screenshot shows a PostgreSQL Query Editor with the following SQL query:

```
8 select * from title_crew writers not in none
9 delete FROM title_crew WHERE writers is null
10 and directors is null and tconst
11 between 'tt0000134' and 'tt0000200';
```

The result shows a message: "DELETE 4. Query returned successfully in 431 msec."

## VIII. QUERY EXECUTION ANALYSIS – USING EXPLAIN TOOL

Performed indexing on three different queries and observed the change in execution time.

### A. Performed indexing on primaryname column of name\_basics table this reduced the execution time:

Before Indexing:

The screenshot shows a PostgreSQL Query Editor with the following SQL query:

```
1 select * from name_basics where primaryname='Arthur'
2
3 create index
```

The result shows the execution plan for the query:

#	Node	Timings	Rows	Loops
1	→ Gather (actual=7471, 90,993 ro...)	Exclusive: 26.715 ms, Inclusive: 90.993 ms	2	1
2	→ Seq Scan on name_basics... Filter: (primaryname)::text = Ar...	Exclusive: 64.278 ms, Inclusive: 64.278 ms	1	3

Indexing:

The screenshot shows a PostgreSQL Query Editor with the following SQL query:

```
1 select * from name_basics where primaryname='Arthur'
2
3 create index idx1 on name_basics(primaryname)
```

The result shows a message: "CREATE INDEX. Query returned successfully in 7 secs 111 msec."

After Indexing:

The screenshot shows a PostgreSQL Query Editor with the following SQL query:

```
1 select * from name_basics where primaryname='Arthur'
2
3 create index idx1 on name_basics(primaryname)
4
5 select * from name_basics where primaryname='Arthur'
```

The result shows the execution plan for the query:

#	Node	Timings	Rows	Loops
1	→ Index Scan using idx1 on name_ba... Index Cond: (primaryname)::text = A...	Exclusive: 0.028 ms, Inclusive: 0.028 ms	2	1

B. Performed indexing on originaltitle, averagerating and genre columns of title\_basics this greatly reduced the execution time of the SELECT Query 5:

Before Indexing:

#	Node	Timings	Rows	Loops
		Exclusive	Inclusive	Actual
1.	→ Hash Right Join (actual=108.58... Hash Cond: ((title_ratings.tconst) = title_basics.tconst))	81.647 ms	236.983 ms	2669
2.	→ Seq Scan on title_ratings as...	50.498 ms	50.498 ms	445783
3.	→ Hash (actual=107.705... Buckets: 4096 Batches: 1 Memo)	1.241 ms	107.839 ms	2669
4.	→ Gather (actual=3.122... Seq Scan on title_basics as... Filter: (genres ~> 'Animation') Rows Removed by Filter: 890)	20.856 ms	106.598 ms	2669

Indexing:

#	Node	Timings	Rows	Loops
		Exclusive	Inclusive	Actual
1.	→ Hash Right Join (actual=1.891... Hash Cond: ((title_ratings.tconst) = title_basics.tconst))	35.506 ms	64.359 ms	2669
2.	→ Seq Scan on title_ratings as...	25.194 ms	25.194 ms	445783
3.	→ Hash (actual=1.596... Buckets: 4096 Batches: 1 Memo)	0.287 ms	1.597 ms	2669
4.	→ Bitmap Heap Scan on title_basics as... Filter: (genres ~> 'Animation') Rows Removed by Filter: 890 Heap Blocks: exact=1190	1.029 ms	1.31 ms	2669
5.	→ Bitmap Index Scan on title_basics_genres_idx	0.281 ms	0.281 ms	2669

After Indexing:

#	Node	Timings	Rows	Loops
		Exclusive	Inclusive	Actual
1.	→ Hash Right Join (actual=1.891... Hash Cond: ((title_ratings.tconst) = title_basics.tconst))	35.506 ms	64.359 ms	2669
2.	→ Seq Scan on title_ratings as...	25.194 ms	25.194 ms	445783
3.	→ Hash (actual=1.596... Buckets: 4096 Batches: 1 Memo)	0.287 ms	1.597 ms	2669
4.	→ Bitmap Heap Scan on title_basics as... Filter: (genres ~> 'Animation') Rows Removed by Filter: 890 Heap Blocks: exact=1190	1.029 ms	1.31 ms	2669
5.	→ Bitmap Index Scan on title_basics_genres_idx	0.281 ms	0.281 ms	2669

C. Performing indexing on birthYear and deathYear column of name\_basics table this also reduces the runtime of the query:

Before Indexing:

#	Node	Timings	Rows	Loops
		Exclusive	Inclusive	Actual
1.	→ Gather (actual=0.919... Seq Scan on name_basics as... Filter: ((birthYear > 1950) AND (deathYear < 2004)) Rows Removed by Filter: 34885	37.36 ms	105.745 ms	1914
2.	→ Seq Scan on name_basics as...	68.386 ms	68.386 ms	638

Indexing:

#	Node	Timings	Rows	Loops
		Exclusive	Inclusive	Actual
1.	→ Gather (actual=0.919... Seq Scan on name_basics as... Filter: ((birthYear > 1950) AND (deathYear < 2004)) Rows Removed by Filter: 34885	37.36 ms	105.745 ms	1914
2.	→ Seq Scan on name_basics as...	68.386 ms	68.386 ms	638

After Indexing:

#	Node	Timings	Rows	Loops
		Exclusive	Inclusive	Actual
1.	→ Bitmap Heap Scan on name_basics as... Recheck Cond: ((deathYear < 2004) AND (birthYear > 1950)) Heap Blocks: exact=1759	0.385 ms	0.071 ms	1914
2.	→ Bitmap AND (actual=8.048... Index Cond: (deathYear < 2004))	0.385 ms	0.071 ms	0
3.	→ Bitmap Index Scan on name_basics_deathYear_idx	3.976 ms	3.976 ms	81232
4.	→ Bitmap Index Scan on name_basics_birthYear_idx	3.688 ms	3.688 ms	88092

## IX. REFERENCES

<https://www.postgresql.org/docs/>

<https://www.w3schools.blog/postgresql-tutorial>

## X. CONTRIBUTION

Task	Task assigned to
Creation of tables	Anudeep
Insertion of data observing nulls	Mahalakshmi
Insertion of data observing nulls	Harinee
select queries(3)	Anudeep
select queries(3)	Mahalakshmi
select queries(3)	Harinee
ER Diagram	Mahalakshmi
Indexing	Harinee
Indexing	Mahalakshmi
Indexing	Anudeep
BCNF	Harinee
Update & delete	Anudeep