IMDB Movie Database

Slytherin

1st Aryan Saini
asaini2
email address - asaini2@buffalo.edu

2nd Divyansh Chopra
dchopra2
email address - dchopra2@buffalo.edu

3rd Pragati Nagar
pragatin
email address - pragatin@buffalo.edu

***Abstract*—For our project we have used the IMDb data. It contains detailed information about movies/TV shows and information related to that. We have taken the data from the infamous IMDb website. The downloaded database contains 7 datasets that contained information about movies and tv-shows till the year 2017 zipped in tsv.tar format. We converted that into csv using excel. The basic idea behind this project is to allow user to make an informed decision on which tv-show/movie to watch and solve various users queries about their favorite shows/movies. Some of the columns in the dataset includes actors, directors, writers, cast, crew, ratings, revenue, release dates, languages, profession, characters, production, genres and many other. Viewers look for the ratings or the cast involved, and some other common details related to the tv-shows/movies they want to watch. This analysis helps the viewers with the information needed by them to choose a tv-show or movie to watch from.**

***Index Terms*—IMDb, Actors, Job, Vote Counts, Genres, Ratings.**

I. Introduction

Main steps involved in the project:

- Analyzing the dataset
- Create tables and schemas
- Define Primary and Foreign key constraints
- Load the data
- Normalizing data
- Define Primary and Foreign key constraints
- Entity relation diagram (ER diagram)
- Getting results using simple and complex queries
- UI creation that executes queries and returns the result

II. Dataset Information

Initial 7 tables information is as mentioned below:

1) Info_title: This relation consists of details about the titles. Below is the detailed information regarding the attributes of the relation.
   - titleId (VARHCAR): Alphanumeric unique identifier of the title.
   - ordering (INTEGER): Uniquely identify row for each titled.
   - title (VARHCAR): Title of each show or movie.
   - Region (VARHCAR): The region corresponding to each title.
   - Language (VARCHAR): The language corresponding to each title.
   - Types (VARCHAR): Enumerated set of attributes for this alternative title. Such as “Orignal”, “dvd” etc.
   - Attributes (VARCHAR): Alternate terms to describe the title.
   - isOriginalTitle (BOOLEAN): 0 for not original, 1 for original title.

2) Episode_title: This relation consists of details about the TV episode information.
   - tconst (VARCHAR): Alphanumeric unique identifier of episode.
   - parentTconst (VARCHAR): Alphanumeric unique identifier of the parent TV show.
   - seasonNumber (INTEGER): Season number corresponding to the episode.
   - episodeNumber (INTEGER): Episode number of the TV show.

3) Crew_title: This relation consists of details of the director, writer.
   - Director (array of nconsts): Determines director(s) corresponding to a title.
   - Writers (array of nconsts): writer(s) corresponding to a title.

4) Basic_title: This relation consists of information about the title being related to a TV show or movie.
   - tconst (VARCHAR): Alphanumeric unique identifier of the title.
   - titleType (VARCHAR): Determines whether the title is of a TV show, TV episode, movie etc.
   - primaryTitle (VARCHAR): Determines the most popular/frequent title used by the content makers around the time of release.
   - originalTitle (VARCHAR): Determines the Original title corresponding to the original language.
   - isAdult (BOOLEAN): Determine whether the content is for adults (1) or not for adults (0).
   - startYear (INTEGER): Determines the release year of a title.
   - endyear (INTEGER): Determines the termination year of the title.
   - runtime (INTEGER): Runtime of the title.
   - genre (VARCHAR): Determines the genre of the title.

5) Principal_crew_title: This relation consists of information about the cast and crew of the titles.

- tconst (VARCHAR): Alphanumeric unique identifier of the title.
- ordering (INTEGER): Uniquely identify row for each titleId.
- nconst (VARCHAR): alphanumeric unique identifier of the name/person.
- category (VARCHAR): Determines the job category of a person involved in title.
- job (VARCHAR): Determines the job title.
- characters (VARCHAR): Determines the name of the character played in a title.

6) Rating_title: This relation consists of information about the IMDB ratings and votes by the audience for different titles.

- tconst (VARCHAR): Alphanumeric unique identifier of the title
- averageRating (INTEGER): Determines the average weighted of all the individual ratings.
- numVotes (INTEGER): Determines the votes received by the title.

7) Crew_information_basic: This relation consists of information about the crew member corresponding to a specific movie, TV show.

- nconst (VARCHAR): alphanumeric unique identifier of the name/person.
- primaryName (VARCHAR): Determines the name of the person most often credited.
- birthYear (INTEGER): Determines the birth year of the corresponding crew member.
- deathYear (INTEGER): Determines the death year of the corresponding crew member,
- primaryProfession (VARCHAR): Determines the top 3 profession.
- knownForTitles (array of tconsts): Determines the titles the person is known for.fp

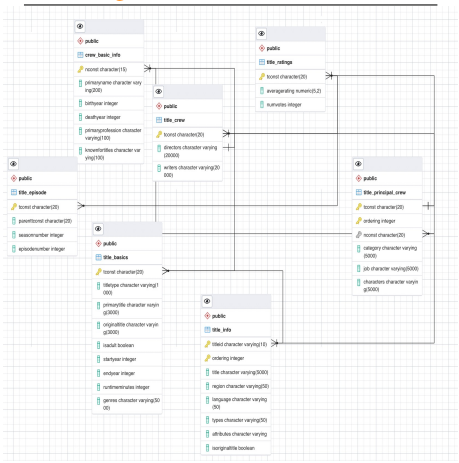
Before normalization constraints of the tables

Table Name	Primary Key
crew_basic_info	nconst
title_basics	tconst
title_crew	tconst
title_episode	tconst
title_info	titleid, ordering
title_principal_crew	tconst, ordering
title_rating	tconst

III. Entity-Relationship (ER) Diagram:

Our initial ER diagram that is of the non-normalized data looks like as shown below:

ER -Diagram of Non-normalized data



IV. Logical Schema:

After analysing the ER diagram above and data in the tables, we concluded that we need to normalize the tables further.

After creating ER diagram, we normalized our database and obtained the logical schema which can be shown and explained below:

- We found that columns such as genre, directors were not in 1NF and we had to change that.
- Considering the 2NF and 3NF we split the 7 tables into 13 tables, which are explained in detail below.

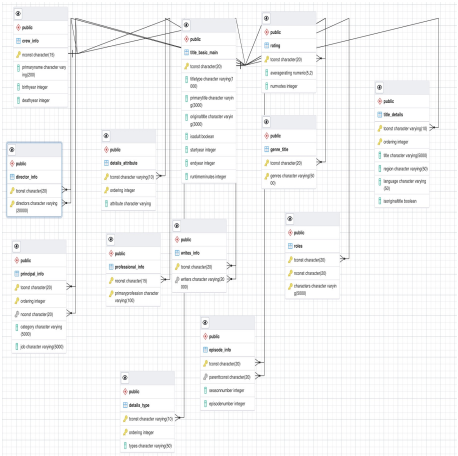
After Normalization constraints of the new tables created.

1) title_basic_main PRIMARY key (tconst)
2) genre_title PRIMARY KEY (tconst,genres) FOREIGN KEY (tconst) REFERENCES public.title_basic_main(tconst) NOT VALID;
3) details_type PRIMARY KEY (tconst,ordering) FOREIGN KEY (tconst) REFERENCES public.title_basic_main(tconst) NOT VALID;
4) details_attribute PRIMARY KEY (tconst,ordering) FOREIGN KEY (tconst) REFERENCES public.title_basic_main(tconst) NOT VALID;
5) rating PRIMARY KEY(tconst) FOREIGN KEY (tconst) REFERENCES public.title_basic_main(tconst) NOT VALID;
6) roles PRIMARY KEY (tconst, nconst, characters) FOREIGN KEY (tconst) REFERENCES public.title_basic_main(tconst) NOT VALID;
7) director_info PRIMARY KEY (tconst, directors) FOREIGN KEY (directors) REFERENCES public.crew_info(nconst) NOT VALID; FOREIGN KEY (tconst) REFERENCES public.title_basic_main(tconst) NOT VALID;
8) writes_info PRIMARY KEY (tconst) FOREIGN KEY (tconst) REFERENCES public.title_basic_main(tconst)

NOT VALID; FOREIGN KEY (writers) REFERENCES public.crew_info(nconst) NOT VALID;

9) principal_info PRIMARY key (tconst,ordering) FOREIGN KEY (nconst) REFERENCES public.crew_info(nconst) NOT VALID; FOREIGN KEY (tconst) REFERENCES public.title_basic_main(tconst) NOT VALID;
10) crew_info PRIMARY KEY (nconst)
11) professional_info PRIMARY KEY (nconst,primaryprofession) FOREIGN KEY (nconst) REFERENCES public.crew_info(nconst) NOT VALID;
12) episode_info PRIMARY KEY (tconst) FOREIGN KEY (parenttconst) REFERENCES public.title_basic_main(tconst) NOT VALID; FOREIGN KEY (tconst) REFERENCES public.title_basic_main(tconst) NOT VALID;
13) title_details PRIMARY KEY (tconst, ordering) FOREIGN KEY (tconst) REFERENCES public.title_basic_main(tconst) NOT VALID;

ER -Diagram of Normalized data



V. IMDb Relations – FD's (1NF, 2NF, 3NF, BCNF)

The functional dependencies and normalization for each relation in our database is explained below: There were certain columns such as genre and directors that included comma separated values. For such columns we normalized them to 1NF by separating those values into multiple rows.

- In order to explain the validity of database in terms of Normalization, we will explore the following functional dependencies.
- Functional Dependencies of the Relation is:
  nconst $\rightarrow$ primaryname, birthyear, deathyear.
- We observe that the relation does not consist of multi valued attributes and hence we can conclude that the relation is in 1NF.
- We observe that primary key of the above functional dependency is nconst and it is also the primary attribute. primaryname, birthyear, deathyear are the non- prime attributes therefore we conclude that the relation is in 2NF since there are no no partial dependencies present.
- We observe that there are no transitive dependencies between prime and non prime attributes therefore the relation is in 3NF.
- We observe that LHS attribute is a super therefore in the mentioned functional dependency the attribute set is a superkey and we conclude that the relation is in BCNF.

VI. Creation and implementation of IMDb database

Information about queries involved:

- Create table queries are in the Create.sql file.
- Load data queries are in the Load.sql file.
- Other queries are in the Queries.sql file.
- CSV formatted data files in the .dat file.

Queries to load the data into the database

- copy crew_basic_info FROM '/Users/aryansaini/Documents/EASSemester2/DataModelsandQueryLanguage/Project/DMQL_FINAL_PROJECT /Data_csv_project/name_basic_info.csv' with csv header;
- copy title_basics FROM '/Users/aryansaini/Documents/EASSemester2/DataModelsandQueryLanguage/Project/DMQL_FINAL_PROJECT /Data_csv_project/title_basics.csv' with csv header;
- copy title_crew FROM '/Users/aryansaini/Documents/EASSemester2/DataModelsandQueryLanguage/Project/DMQL_FINAL_PROJECT /Data_csv_project/title_crew.csv' with csv header;
- copy title_episode FROM '/Users/aryansaini/Documents/EASSemester2/Data ModelsandQueryLanguage/Project/DMQL_FINAL_PROJECT /Data_csv_project/title_episode.csv' with csv header;
- copy title_info FROM '/Users/aryansaini/Documents/EASSemester2/DataModelsandQueryLanguage/Project/DMQL_FINAL_PROJECT /Data_csv_project /title_info.csv' with csv header;
- copy title_principal_crew FROM '/Users/aryansaini/Documents/EASSemester2/DataModelsandQueryLanguage/Project/DMQL_FINAL_PROJECT /Data_csv_project/title_principal_crew.csv' with csv header;
- copy title_ratings FROM '/Users/aryansaini/Documents/EASSemester2/DataModelsandQueryLanguage/Project/DMQL_FINAL_PROJECT /Data_csv_project/title_ratings.csv' with csv header;

VII. SQL Queries

1) List the distinct types of professions in the database and the count of each type of profession?

SELECT a.category FROM principal_info as a GROUP BY a.category ORDER BY a.category ASC;

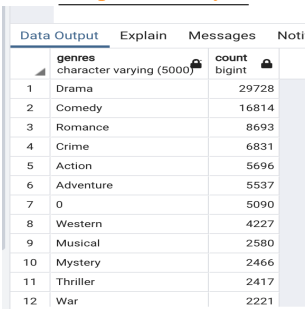
Output - Query 1

Data Output | Explain | Messages

	category character varying (5000)
1	actor
2	actress
3	archive_footage
4	cinematographer
5	composer
6	director
7	editor
8	producer
9	production_designer
10	self
11	writer

2) What are the different type of genres? How many movies are there in each genre?
SELECT G.genres, COUNT(G.genres) AS Count FROM genre_title AS G, title_basic_main AS T WHERE T.tconst = G.tconst AND T.titletype = 'movie' GROUP BY genres ORDER BY Count DESC;

Output - Query 2

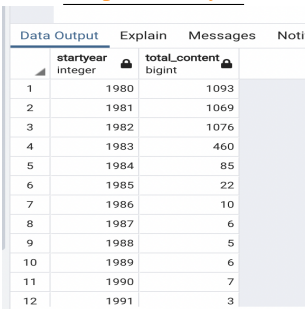


Data Output | Explain | Messages | Noti

	genres character varying (5000)	count bigint
1	Drama	29728
2	Comedy	16814
3	Romance	8693
4	Crime	6831
5	Action	5696
6	Adventure	5537
7	0	5090
8	Western	4227
9	Musical	2580
10	Mystery	2466
11	Thriller	2417
12	War	2221

3) What is the count of movies, short made in each year from 1980 to 2005?
SELECT a.startYear, COUNT(*) AS total_content FROM title_basic_main AS a WHERE a.titleType IN ('movie','short') GROUP BY a.startYear HAVING a.startYear BETWEEN 1980 AND 2005 ORDER BY a.startYear ASC;

Output- Query 3



Data Output | Explain | Messages | Noti

	startyear integer	total_content bigint
1	1980	1093
2	1981	1069
3	1982	1076
4	1983	460
5	1984	85
6	1985	22
7	1986	10
8	1987	6
9	1988	5
10	1989	6
11	1990	7
12	1991	3

4) List the content(any type) whose duration is longer than 2 hours?
SELECT tconst , runtimeMinutes, titleType, primaryTitle FROM Title_Basic_main WHERE runtimeMinutes >(2*60)
ORDER BY runtimeMinutes DESC, titleType ASC;

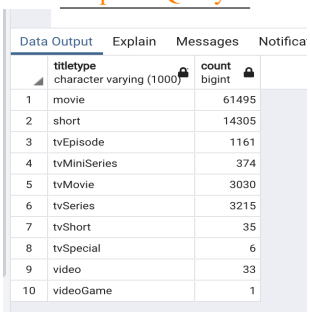
Output - Query 4

Data Output | Explain | Messages | Notifications

	tconst [PK] character (10)	runtimeminutes integer	titletype character varying (1000)	primarytitle character varying (3000)
1	tt0804852	1428	movie	The Hazards of Helen
2	tt0370557	1335	tvMiniSeries	The Pallisers
3	tt0376993	1256	tvMiniSeries	Centennial
4	tt0880396	931	tvMiniSeries	Berlin Alexanderplatz
5	tt0869654	890	tvSeries	War & Peace
6	tt0859212	883	tvMiniSeries	The Winds of War
7	tt0376878	840	tvMiniSeries	Roots: The Next Generations
8	tt0377310	780	tvMiniSeries	Flambards
9	tt0381846	780	tvMiniSeries	Cosmos
10	tt0381870	780	tvSeries	Los pazos y las sombras
11	tt0377846	780	tvSeries	Love for Lydia
12	tt0374369	770	tvSeries	My Uncle Napoleon

5) List the distinct types of titles present in the database.
SELECT T.titletype, COUNT(*) FROM title_basic_main AS T GROUP BY T.titletype ORDER BY T.titletype ASC;

Output - Query 5

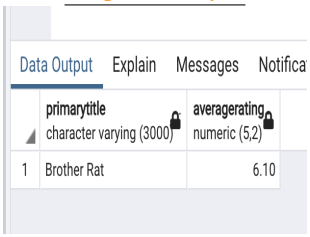


Data Output | Explain | Messages | Notifica

	titletype character varying (1000)	count bigint
1	movie	61495
2	short	14305
3	tvEpisode	1161
4	tvMiniSeries	374
5	tvMovie	3030
6	tvSeries	3215
7	tvShort	35
8	tvSpecial	6
9	video	33
10	videoGame	1

6) What is the mean rating of the title "Brother Rat"
SELECT t.primaryTitle, R.averageRating FROM title_basic_main AS t, rating AS R WHERE t.tconst = R.tconst AND t.titleType = 'movie' AND t.primaryTitle like 'Brother Rat';

Output -Query 6



Data Output | Explain | Messages | Notifica

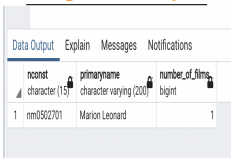
	primarytitle character varying (3000)	averagerating numeric (5,2)
1	Brother Rat	6.10

7) **Using Views**
Which actor played the role of 'Isabelle' in a 'short' titletype How many times have they played that role
CREATE OR REPLACE VIEW
Q1(nconst,primaryname,number_of_films) AS
SELECT N.nconst, N.primaryname, COUNT(*) AS number_of_films
FROM crew_info AS N, roles AS H, title_basic_main AS T
WHERE H.characters LIKE 'Isabelle'
AND T.titletype LIKE 'short'
AND T.tconst = H.tconst

>

```
AND N.nconst = H.nconst
GROUP BY N.nconst;
SELECT * FROM Q1;
```

Output -Query 7



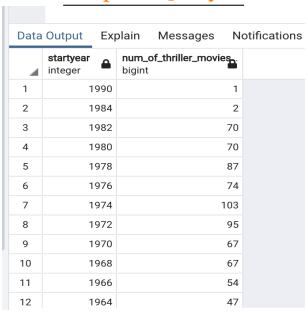
	nconst character (15)	primaryname character varying (200)	number_of_films bigint
1	nm0502701	Marion Leonard	1

8) **Using Views**

How many Thriller of titletype (movies, short) are made in even years?

```
CREATE OR REPLACE VIEW Q2(StartYear,Num_of_Thriller_movies)
AS SELECT T.startyear, COUNT(DISTINCT T.tconst)
AS Num_of_Thriller_movies
FROM title_basic_main AS T, genre_title AS G
WHERE T.tconst = G.tconst
AND G.genres = 'Thriller'
AND T.titletype IN ('movie','short')
AND (T.startyear % 2) = 0
GROUP BY T.startyear
ORDER BY T.startyear DESC;
SELECT * FROM Q2;
```

Output -Query 8



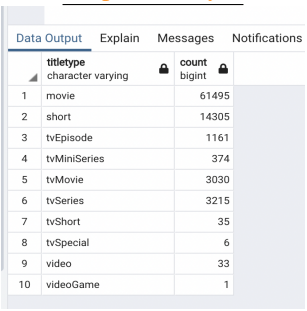
	startyear integer	num_of_thriller_movies bigint
1	1990	1
2	1984	2
3	1982	70
4	1980	70
5	1978	87
6	1976	74
7	1974	103
8	1972	95
9	1970	67
10	1968	67
11	1966	54
12	1964	47

9) **Function**

Creating a function below that List the distinct types of titles present in the database.

```
CREATE OR REPLACE FUNCTION distinct_title()
RETURNS table(
titletype character varying(1000),
count bigint
)
AS $$
BEGIN
RETURN QUERY SELECT T.titletype, COUNT(*)
FROM title_basic_main AS T
GROUP BY T.titletype
ORDER BY T.titletype ASC;
END; $$
LANGUAGE plpgsql;
# Calling the function
select * from distinct_title()
```

Output -Query 9



	titletype character varying	count bigint
1	movie	61495
2	short	14305
3	tvEpisode	1161
4	tvMiniSeries	374
5	tvMovie	3030
6	tvSeries	3215
7	tvShort	35
8	tvSpecial	6
9	video	33
10	videoGame	1

10) **Trigger**

Below will create a new table avg_rating_percentage, create a trigger which will insert new values of the average rating into the avg_rating table. create table avg_rating_percentage(percentage numeric(5,5));

```
CREATE OR REPLACE FUNCTION
insert_avg_rating_percentage()
RETURNS trigger AS
$$
BEGIN
INSERT INTO avg_rating_percentage(percentage)
VALUES(NEW.percentage);
RETURN NEW;
END;
$$
LANGUAGE 'plpgsql';
CREATE TRIGGER percentage_trigger
AFTER INSERT
ON avg_rating_percentage
FOR EACH ROW
EXECUTE PROCEDURE
insert_avg_rating_percentage();
```

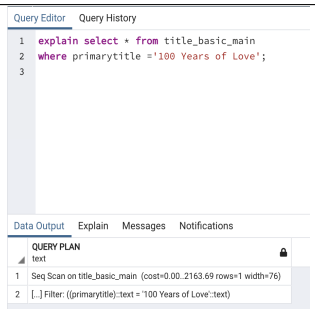
VIII. INDEXING

While running some queries we were facing issues with the execution time. Although our data is not huge but still the execution time to run queries was significant. For that we did indexing on the title_basic_main table The difference in the execution time before and after indexing

Before indexing

```
explain select * from title_basic_main
where primarytitle ='100 Years of Love';
```

Execution time before indexing



Query Editor | Query History

```
1 explain select * from title_basic_main
2 where primarytitle ='100 Years of Love';
3
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Seq Scan on title_basic_main (cost=0.00..2163.69 rows=1 width=76)
2	[...] Filter: ((primarytitle)::text = '100 Years of Love'::text)

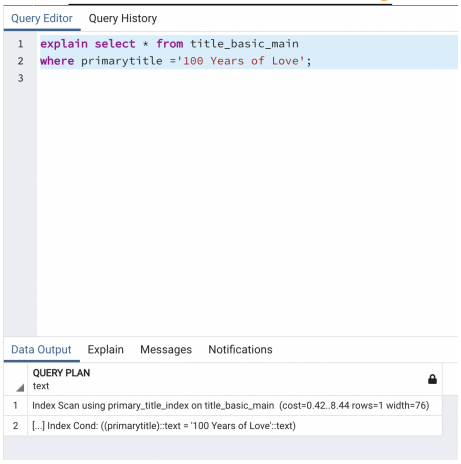
Creating an index

```
CREATE INDEX primary_title_index
ON title_basic_main(primarytitle);
```

After indexing

```
explain select * from title_basic_main
where primarytitle ='100 Years of Love';
```

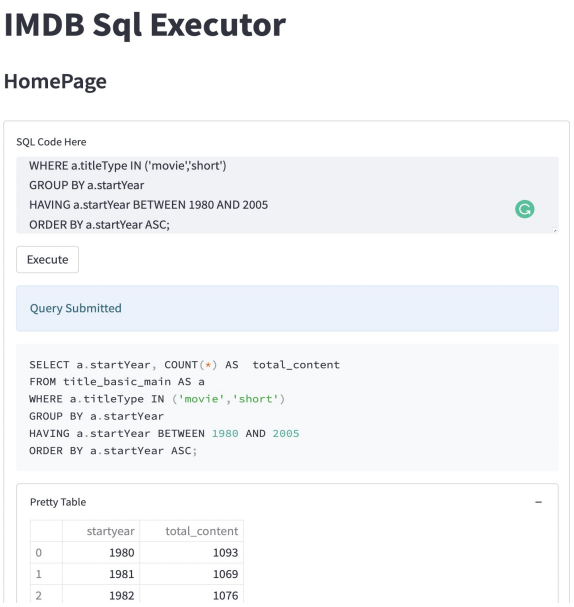
Execution time after indexing



IX. Visualization

In order to visualize our database and have a workplace to execute our queries into, we built our front end using Python and Streamlit lab. We built a SQL Playground which runs in the web browser. The user has the option to execute any valid query and see it's result in the web page. We provide results in two manners: Pretty Table and JSON formatting. Furthermore, we also display the last query which was run so that the user can keep a track of them.

Execution time after indexing



X. Contribution

Team - Slytherin

Team members as follows:

- Aryan Saini (asaini2)
- Divyansh Chopra (dchopra2)
- Pragati Nagar (pragatin)

Workflow includes following steps:

- Downloading and loading data files.
- Created 7 tables initially from the csv files.
- Designed logical schema and ER diagram for 7 tables.
- Further normalized the data and created 13 new tables.
- Loaded data into those normalized tables.
- Set Primary and Foreign key constraints.
- Logical schema and ER diagram for normalized data.
- Indexing
- Simplex and complex queries to test the data.
- UI creation

Team Member	**Contribution percentage(%)**
Aryan Saini	33.33
Divyansh Chopra	33.33
Pragati Nagar	33.33