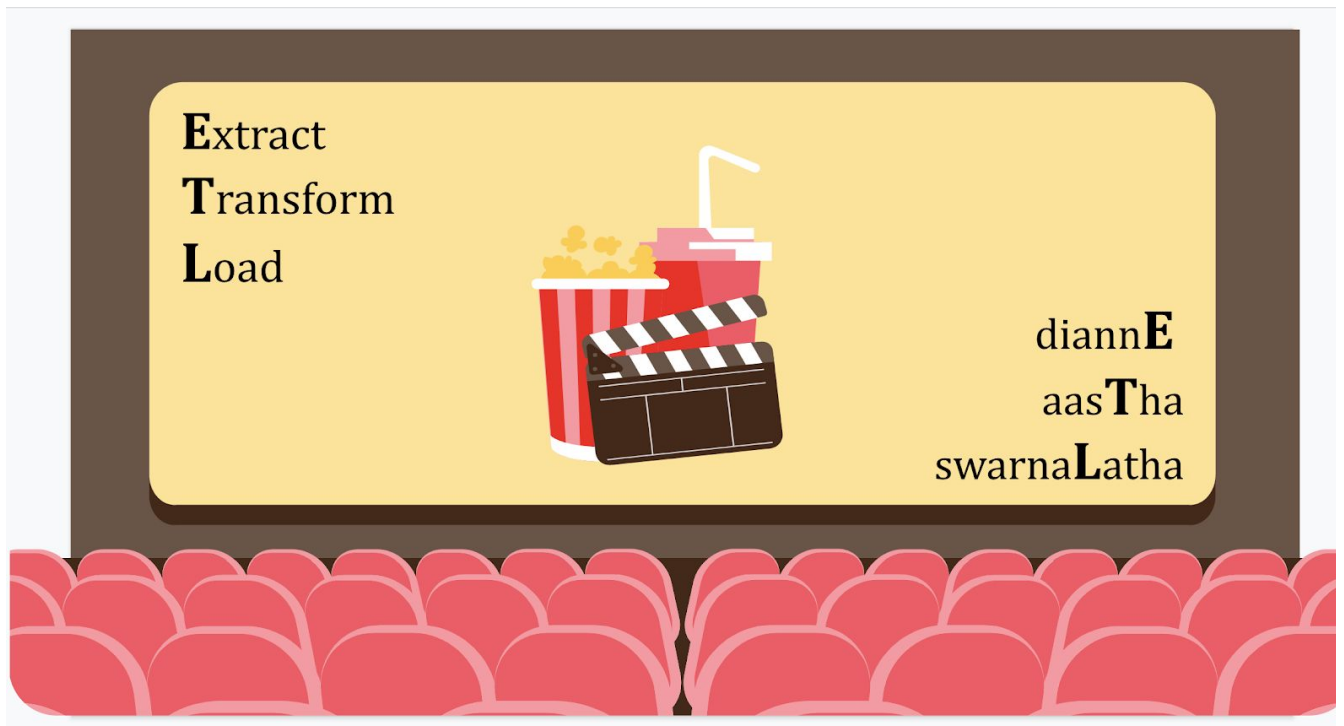


# PROJECT REPORT



Team Members:

Aastha Arora  
Dianne Jardinez  
Swarna Latha

## Summary

The objective of this project was to extract data from websites and available APIs. The following datasets were then transformed by cleaning, joining, and filtering into seven tables. The object-relational database (PostgreSQL) was used to load the datasets into pgAdmin.

<b>Summary</b>	<b>1</b>
<b>Extract</b>	<b>3</b>
IMDb Website	3
OMDb API	3
Utelly API	4
uNoGS API	5
Google Search Engine	7
<b>Transform</b>	<b>9</b>
Tables	9
top_imdb	9
movie	10
movie_actor	11
movie_director	12
utelly_streaming_service	13
google_streaming_service	14
utelly_movie_streaming	15
google_movie_streaming	16
netflix_movie	17
<b>Load</b>	<b>18</b>
ERD Diagram	18
ERD Text	19
SQL Schema	20
Sample Queries	22
Query: Joining movie table with google_movie_streaming table	22
Query: Specific movie title, streaming service availability, price	22

## Extract

---

The following Data Sources were used below:

### IMDb Website

- Method: Web Scraping
- Link: [https://www.imdb.com/chart/top/?ref\\_=nv\\_mv\\_250](https://www.imdb.com/chart/top/?ref_=nv_mv_250)
- Used for: Collecting the Top 250 IMDB rated movie list
- Description: We used the url and pandas code to get the HTML table on the webpage. The data contains a list of top 250 imdb movie titles along with imdb rank and imdb rating.

	Unnamed: 0	Rank & Title	IMDb Rating	Your Rating	Unnamed: 4
0	NaN	1. The Shawshank Redemption (1994)	9.2	12345678910 NOT YET RELEASED Seen	NaN
1	NaN	2. The Godfather (1972)	9.1	12345678910 NOT YET RELEASED Seen	NaN
2	NaN	3. The Godfather: Part II (1974)	9.0	12345678910 NOT YET RELEASED Seen	NaN
3	NaN	4. The Dark Knight (2008)	9.0	12345678910 NOT YET RELEASED Seen	NaN
4	NaN	5. 12 Angry Men (1957)	8.9	12345678910 NOT YET RELEASED Seen	NaN



## Utelly API

- Method: API Extraction
- Link:  
[https://rapidapi.com/utelly/api/utelly?endpoint=apiendpoint\\_3cad787b-ca7b-449a-84b4-23b40d64fd73](https://rapidapi.com/utelly/api/utelly?endpoint=apiendpoint_3cad787b-ca7b-449a-84b4-23b40d64fd73)
- Used For: Collecting streaming options for Top 250 IMDb movies
- Description: Utelly API is an API available on rapidapi.com. A personal account was created to get the api\_key required to request data from this API. The API had limits on the number of requests that we can make in a day.

The endpoint used was GET/idlookup and it took the imdb\_id to return a JSON file for each movie. We used the imdb\_id obtained from the OMDb API. 250 requests were made using this API to obtain 250 json files, one for each movie. The json files contained the streaming options that are available for a particular movie. The returned json files were searched to look for the information that we needed and it was saved in a pandas dataframe. The information was collected and saved in such a way that we could get the dataframe in first normal form.

	IMDb ID	Title	Streaming Service	Streaming URL
0	tt0111161	The Shawshank Redemption	Netflix	<a href="https://www.netflix.com/title/70005379">https://www.netflix.com/title/70005379</a>
1	tt0111161	The Shawshank Redemption	Google Play	<a href="https://play.google.com/store/movies/details/T...">https://play.google.com/store/movies/details/T...</a>
2	tt0111161	The Shawshank Redemption	Amazon Instant Video	<a href="https://www.amazon.com/gp/product/B001EBV0P8?c...">https://www.amazon.com/gp/product/B001EBV0P8?c...</a>
3	tt0111161	The Shawshank Redemption	iTunes	<a href="https://itunes.apple.com/us/movie/the-shawshan...">https://itunes.apple.com/us/movie/the-shawshan...</a>
4	tt0068646	The Godfather	Google Play	<a href="https://play.google.com/store/movies/details/T...">https://play.google.com/store/movies/details/T...</a>
...	...	...	...	...
694	tt0103639	Aladdin	Google Play	<a href="https://play.google.com/store/movies/details/A...">https://play.google.com/store/movies/details/A...</a>
695	tt0103639	Aladdin	DisneyPlusIVAUS	<a href="https://www.disneyplus.com/movies/aladdin-1992...">https://www.disneyplus.com/movies/aladdin-1992...</a>
696	tt2338151	PK	Netflix	<a href="https://www.netflix.com/title/70303496">https://www.netflix.com/title/70303496</a>
697	tt0094625	Akira	Hulu	<a href="https://www.hulu.com/watch/64a5a8d0-1406-4178-...">https://www.hulu.com/watch/64a5a8d0-1406-4178-...</a>
698	tt0050613	Throne of Blood	iTunes	<a href="https://itunes.apple.com/us/movie/throne-of-bl...">https://itunes.apple.com/us/movie/throne-of-bl...</a>

## uNoGS API

- Method: API Extraction
- Link: <https://rapidapi.com/unogs/api/unogs/endpoints>
- Used For: Collecting movies on Netflix in released in the United States which have an IMDb rating greater than or equal to 7
- Description: uNoGS (unofficial Netflix online Global Search) allows anyone to search the global Netflix catalog. Netflix no longer provides an API for its data. So an alternative Netflix API was used. This API is available on rapidapi.com. A personal account had to be created on rapidapi.com to get the api\_key needed to extract data from this API.

The API provides several API endpoints. Each of the endpoints were analysed using sample requests to figure out the API that could be most useful for the purpose of the project. The endpoint used was GET Advanced Search. This endpoint provided an option to build an advanced custom query. Search could be based on a number of different parameters including netflix country list, movie/series type, genre, subtitles, audio, imdb rating, dates on which the movie became available on Netflix and some other parameters.

The parameters that we used for this search were

1. Movie or Series: We used to code only to get a list of movies
2. Country code: We used the code to get only the movies that are available in the United States (country code: 78 on API found using a different query)
3. IMDb rating: We used to code to get only the movies that had the IMDb rating between 7 and 10.
4. Movie year range was year 1900 to 2020 to get almost all the movies that satisfy the above conditions.
5. No filtering was done for genre, subtitles, audio, etc.

The API has a limit to return only 100 results per page per request. The return results of the query returned around 925 total results. Hence the request was made 10 times (ie, to get 10 pages of result) to get all the results that satisfy all the conditions listed above. The data obtained was in json format. The data on all the pages was combined in a json file and was loaded in a pandas dataframe using `pd.read_json()`

Code reference jupyter notebook: `netflix_high_imdbRated(uNoGS api)`

	netflixid	title	image	synopsis	rating	type	released	runtime	largeimage
0	80192064	Luciano Meller: Infantiloide	<a href="http://occ-0-2851-38.1.nflxso.net/dnm/api/v6/e...">http://occ-0-2851-38.1.nflxso.net/dnm/api/v6/e...</a>	Argentina's Luciano Meller emphasizes the...	9.6	movie	2018	1h6m	
1	81206389	Oththa Seruppu Size 7	<a href="https://occ-0-2851-38.1.nflxso.net/dnm/api/v6/...">https://occ-0-2851-38.1.nflxso.net/dnm/api/v6/...</a>	Taken into custody, a murder suspect's the...	9.4	movie	2019	1h43m	
2	553500	The Good, the Bad and the Ugly	<a href="https://occ-0-1091-300.1.nflxso.net/dnm/api/v6...">https://occ-0-1091-300.1.nflxso.net/dnm/api/v6...</a>	While the Civil War rages between the Union an...	8.8	movie	1966	2h58m	<a href="http://cdn1.nflximg">http://cdn1.nflximg</a>



## Google Search Engine

- Method: Web Scraping
- Link: A custom link was generated for each movie

The titles of the movies (top 250 imdb movies) were converted to lowercase and the space between any two words in the movie title was replaced by '+'. Also the string '+watch+movie' was appended at the end of the url to get the data that we need.

- Used for: Collecting viewing Streaming Service availability and price.
- Description: Beautiful Soup library was used to scrape the data using css class names. Google has a limitation on the number of requests that can be made per hour. It was found through 'Google Search' that this number is 8 requests per hour per IP address. There is a possibility that if the number of requests exceed 10, Google will block the IP address. So scraping the data involved a lot of effort in running the code in sets of 8, multiple times during the day. The data obtained was transformed in pandas data frame.

Title	Streaming On	Price
The Shawshank Redemption	YouTube	3.99
The Shawshank Redemption	iTunes	3.99
The Shawshank Redemption	Google Play Movies & TV	3.99
The Shawshank Redemption	Vudu	3.99
The Shawshank Redemption	Amazon Prime Video	3.99
The Shawshank Redemption	Philo	Subscription
The Godfather	YouTube	2.99
The Godfather	Google Play Movies & TV	2.99
The Godfather	Vudu	2.99
The Godfather	Amazon Prime Video	2.99
The Godfather: Part II	YouTube	2.99
The Godfather: Part II	Google Play Movies & TV	2.99
The Godfather: Part II	Vudu	2.99
The Godfather: Part II	Amazon Prime Video	2.99
The Dark Knight	YouTube	3.99
The Dark Knight	Google Play Movies & TV	3.99

## Transform

- Data extracted were formatted in CSV and JSON files
- The data formatted files were manipulated in pandas to clean, join, and filter seven tables

## Tables

top\_imdb

	imdb_id	title
0	tt0111161	The Shawshank Redemption
1	tt0068646	The Godfather
2	tt0071562	The Godfather: Part II
3	tt0468569	The Dark Knight
4	tt0050083	12 Angry Men
...	...	...
245	tt0064115	Butch Cassidy and the Sundance Kid
246	tt0103639	Aladdin
247	tt2338151	PK
248	tt0094625	Akira
249	tt0050613	Throne of Blood

Transformation steps:

- The data table obtained from the IBDb website link has IMDb rank, Movie title and Year of Release in a single column. These were converted in separate columns to get the list of movies. This list was then used as an input for the OMDb API.
- Displayed just the IMDb ID and movie title from the OMDb API data
- Saved the DataFrame to a csv file

## movie

	imdb_id	imdb_rank	title	year	runtime	rated	imdb_rating	production
0	tt0111161	0	The Shawshank Redemption	1994	142	R	9.3	Columbia Pictures
1	tt0068646	1	The Godfather	1972	175	R	9.2	Paramount Pictures
2	tt0071562	2	The Godfather: Part II	1974	202	R	9.0	Paramount Pictures
3	tt0468569	3	The Dark Knight	2008	152	PG-13	9.0	Warner Bros. Pictures/Legendary
4	tt0050083	4	12 Angry Men	1957	96	Approved	8.9	Criterion Collection

### Transformation steps:

- Displayed IMDb ID, movie title, year, release year, runtime, movie rating, IMDb rating, and production from the OMDb API data
- Reset the index to make the IMDb rank column
- Transformed the decimal significant figure from 1 to 0
- Split the runtime string to only include the number
- Renamed columns of the runtime data frame to drop the unwanted string from runtime
- Renamed the columns of the movie data frame to include only meaningful names
- Combined runtime and movie data frame
- Displayed IMDb ID, movie title, year, release year, runtime, movie rating, IMDb rating, and production from movie data frame
- Saved the DataFrame to a csv file

## movie\_actor

	imdb_id	actor
0	tt0111161	Tim Robbins
1	tt0111161	Morgan Freeman
2	tt0111161	Bob Gunton
3	tt0111161	William Sadler
4	tt0068646	Marlon Brando
...	...	...
972	tt0094625	Tesshō Genda
973	tt0050613	Toshirō Mifune
974	tt0050613	Isuzu Yamada
975	tt0050613	Takashi Shimura
976	tt0050613	Akira Kubo

### Transformation steps:

- Displayed IMDb ID, and actors from the OMDb API data
- Dropped any N/A values
- Creating an actor list to set up a conversion into a DataFrame
- Iterated through actors dataframe to clean up data into a dictionary
- Converted actor list into a Dataframe
- Saved the DataFrame to a csv file

## movie\_director

	imdb_id	director
0	tt0111161	Frank Darabont
1	tt0068646	Francis Ford Coppola
2	tt0071562	Francis Ford Coppola
3	tt0468569	Christopher Nolan
4	tt0050083	Sidney Lumet
...	...	...
265	tt0103639	Ron Clements
266	tt0103639	John Musker
267	tt2338151	Rajkumar Hirani
268	tt0094625	Katsuhiro Ôtomo
269	tt0050613	Akira Kurosawa

### Transformation steps:

- Displayed IMDb ID, and directors from the OMDb API data
- Dropped any N/A values
- Creating a director list to set up a conversion into a DataFrame
- Iterated through directors dataframe to clean up data into a dictionary
- Converted director list into a Dataframe
- Saved the DataFrame to a csv file

## utelly\_streaming\_service

	service_id	service_name
0	ss_1	Amazon Instant Video
1	ss_2	Amazon Prime Video
2	ss_3	AtomTickets
3	ss_4	DisneyPlus
4	ss_5	FandangoMovies
5	ss_6	Google Play
6	ss_7	HBO
7	ss_8	Hulu
8	ss_9	Netflix
9	ss_10	iTunes

### Transformation steps:

- Displayed streaming services from the Utelly API data
- Renamed column header to have meaningful names
- Filtered to see unique values in Streaming service column of dataframe to set up for a unique service id
- Splitting service name column by splitting string
- Grouped Streaming services
- Renamed columns to meaningful names
- Dropping unwanted column that did not include streaming name
- Declared a list for unique service id column
- Created a service id column and added to the service\_id list
- Show service\_id and service\_name of the utelly\_streaming\_Service table
- Saved the DataFrame to a csv file

## google\_streaming\_service

	<b>service_id</b>	<b>service_name</b>
0	gg_1	Amazon Prime Video
1	gg_2	Cinemax
2	gg_3	Crackle
3	gg_4	Disney+
4	gg_5	Google Play Movies & TV
5	gg_6	HBO Now
6	gg_7	Hulu
7	gg_8	Netflix
8	gg_9	Philo
9	gg_10	Showtime
10	gg_11	Sling TV
11	gg_12	Starz
12	gg_13	Tubi
13	gg_14	Vudu
14	gg_15	YouTube
15	gg_16	fuboTV
16	gg_17	iTunes

### Transformation steps:

- Displayed streaming services from the Google web scraped data
- Renamed column header to have meaningful names
- Filtered to see unique values in Streaming service column of dataframe to set up for a unique service id
- Grouped Streaming services
- Renamed columns to meaningful names
- Dropping unwanted column that did not include streaming name
- Declared a list for unique service id column
- Created a service id column and added to the service\_id list
- Show service\_id and service\_name of the google\_streaming\_service table
- Saved the DataFrame to a csv file

## utelly\_movie\_streaming

	imdb_id	service_id
0	tt0111161	ss_9
1	tt0111161	ss_6
2	tt0111161	ss_1
3	tt0111161	ss_10
4	tt0068646	ss_6
...	...	...
694	tt0103639	ss_6
695	tt0103639	ss_4
696	tt2338151	ss_9
697	tt0094625	ss_8
698	tt0050613	ss_10

Transformation steps:

- Displayed streaming services from the Utelly API data
- Transformed streaming service column by splitting string
- Dropped unwanted column that does not include important information for service dataframe
- Renamed columns to meaningful names
- Combined rename dataframe with service dataframe
- Created a blank series for the service\_id column
- Ran a for loop and assigning values to service\_id series
- Inserted new column from the service\_id values
- Showed service id and IMDB id of the utelly\_movie\_streaming table
- Saved the DataFrame to a csv file



## google\_movie\_streaming

	title	google_service_id	price
0	The Shawshank Redemption	gg_17	3.99
1	The Shawshank Redemption	gg_9	3.99
2	The Shawshank Redemption	gg_6	3.99
3	The Shawshank Redemption	gg_16	3.99
4	The Shawshank Redemption	gg_1	3.99
...	...	...	...
1171	Akira	gg_16	2.99
1172	Akira	gg_9	5.99
1173	Akira	gg_7	Subscription
1174	Throne Of Blood	gg_9	3.99
1175	Throne Of Blood	gg_1	3.99

### Transformation steps:

- Displayed streaming services from the Google web scraped data
- Renamed column header to have meaningful names
- Filtered to see unique values in Streaming service column of dataframe to set up for a unique service id
- Created a blank series for the service\_id column
- Ran a for loop and assigning values to service\_id series
- Inserted new column from the service\_id values
- Showed movie title, google service id and price of the google\_movie\_streaming table
- Saved the DataFrame to a csv file

## netflix\_movie

	netflix_id	imdb_id	title
2	553500	tt0060196	The Good, the Bad and the Ugly
3	70131314	tt1375666	Inception
8	20557937	tt0133093	The Matrix
17	60010110	tt0088763	Back to the Future
18	60031884	tt0064116	Once Upon a Time in the West
...	...	...	...
863	70114021	tt1019452	A Serious Man
864	80017021	tt2937898	A Most Violent Year
866	70130432	tt0081494	Return To The 36th Chamber
867	80025390	tt3007512	The Water Diviner
868	80046694	tt4178092	The Gift

Transformation steps:

- Displayed Netflix ID, IMDB ID, and movie title from the Rapid API data
- Renames columns into meaningful names
- Saved the DataFrame to a csv file

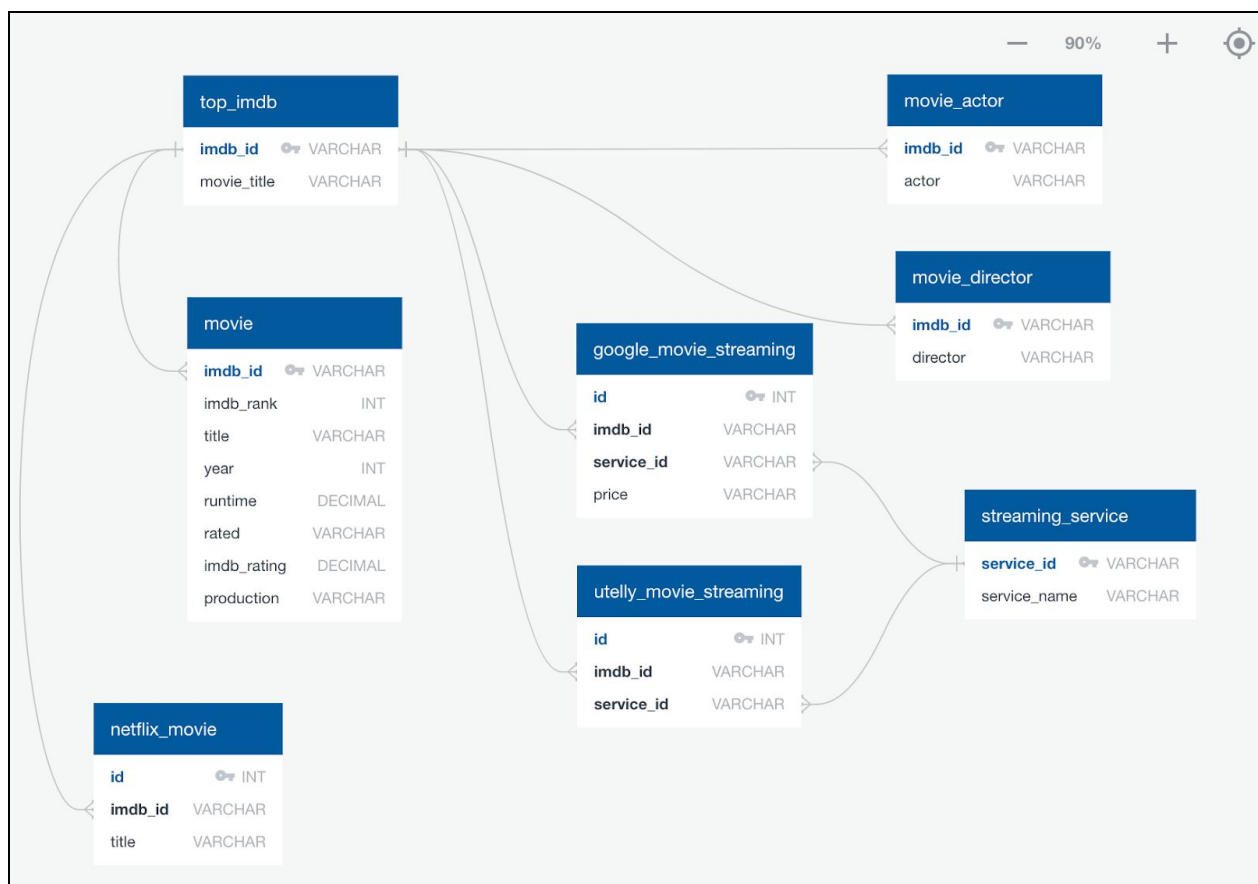
## Load

- The object-relational database (PostgreSQL) was used to load the datasets into pgAdmin
- We selected a relational database as the data was in a structured format

Table Schema was generated and sqlalchemy engine was used to load the data into the SQL tables

- Code Reference: SQL\_Tables.ipynb

## ERD Diagram



## ERD Text

```

top_imdb
-
  imdb_id VARCHAR PK
  movie_title VARCHAR

movie
-
  imdb_id VARCHAR PK FK >- top_imdb.imdb_id
  imdb_rank INT
  title VARCHAR
  year INT
  runtime DECIMAL
  rated VARCHAR
  imdb_rating DECIMAL
  production VARCHAR

movie_actor
-
  imdb_id VARCHAR PK FK >- top_imdb.imdb_id
  actor VARCHAR

movie_director
-
  imdb_id VARCHAR PK FK >- top_imdb.imdb_id
  director VARCHAR

streaming_service
-
  service_id VARCHAR PK
  service_name VARCHAR

google_movie_streaming
-
  title VARCHAR FK >- top_imdb.imdb_id
  service_id VARCHAR FK >- streaming_service.service_id
  price VARCHAR

utelly_movie_streaming
-
  imdb_id VARCHAR FK >- top_imdb.imdb_id
  service_id VARCHAR FK >- streaming_service.service_id

netflix_movie
-
  id INT PK
  imdb_id VARCHAR FK >- top_imdb.imdb_id
  title VARCHAR

```

## SQL Schema

```
DROP TABLE IF EXISTS top_imdb;
DROP TABLE IF EXISTS movie;
DROP TABLE IF EXISTS movie_actor;
DROP TABLE IF EXISTS movie_director;
DROP TABLE IF EXISTS streaming_service;
DROP TABLE IF EXISTS netflix_movie;

create table top_imdb(
    imdb_id VARCHAR not null,
    movie_title VARCHAR not null,
    PRIMARY KEY (imdb_id)
);

create table movie(
    imdb_id VARCHAR not null,
    imdb_rank INT not null,
    title VARCHAR not null,
    year INT not null,
    runtime DECIMAL not null,
    rated VARCHAR not null,
    imdb_rating DECIMAL not null,
    production VARCHAR not null,
    PRIMARY KEY (imdb_id),
    FOREIGN KEY (imdb_id) REFERENCES top_imdb (imdb_id)
);

create table movie_actor(
    imdb_id VARCHAR not null,
    actor VARCHAR not null,
    FOREIGN KEY (imdb_id) REFERENCES top_imdb (imdb_id)
);

create table movie_director(
    imdb_id VARCHAR not null,
    director VARCHAR not null,
    FOREIGN KEY (imdb_id) REFERENCES top_imdb (imdb_id)
);

create table streaming_service(
    service_id VARCHAR not null,
    service_name VARCHAR not null,
    PRIMARY KEY (service_id)
);

create table google_movie_streaming(
    id INT not null,
    imdb_id VARCHAR not null,
```

```
service_id VARCHAR not null,  
price VARCHAR not null,  
PRIMARY KEY (id,imdb_id),  
FOREIGN KEY (imdb_id) REFERENCES top_imdb (imdb_id),  
FOREIGN KEY (service_id) REFERENCES streaming_service (service_id)  
);  
  
create table utelly_movie_streaming(  
id INT not null,  
imdb_id VARCHAR not null,  
service_id VARCHAR not null,  
PRIMARY KEY (id,imdb_id),  
FOREIGN KEY (imdb_id) REFERENCES top_imdb (imdb_id),  
FOREIGN KEY (service_id) REFERENCES streaming_service (service_id)  
);  
  
create table netflix_movie(  
id INT not null,  
imdb_id VARCHAR not null,  
title VARCHAR not null,  
PRIMARY KEY (id),  
FOREIGN KEY (imdb_id) REFERENCES top_imdb (imdb_id)  
);
```

## Sample Queries

### Query: Joining movie table with google\_movie\_streaming table

```

101 select * from movie
102 join google_movie_streaming
103 on lower(movie.title) = lower(google_movie_streaming.title)
104 join google_streaming_service
105 on google_movie_streaming.service_id = google_streaming_service.service_id
106

```

	imdb_id character varying	imdb_rank integer	title character varying	year integer	runtime numeric	rated character varying	imdb_rating numeric
1	tt0050083	5	12 Angry Men	1957	96	Approved	
2	tt0050083	5	12 Angry Men	1957	96	Approved	
3	tt0050083	5	12 Angry Men	1957	96	Approved	
4	tt0050083	5	12 Angry Men	1957	96	Approved	
5	tt0050083	5	12 Angry Men	1957	96	Approved	
6	tt2024544	202	12 Years a Slave	2013	134	R	
7	tt2024544	202	12 Years a Slave	2013	134	R	
8	tt2024544	202	12 Years a Slave	2013	134	R	
9	tt2024544	202	12 Years a Slave	2013	134	R	
10	tt2024544	202	12 Years a Slave	2013	134	R	
11	tt0062622	90	2001: A Space Odyssey	1968	149	G	
12	tt0062622	90	2001: A Space Odyssey	1968	149	G	
13	tt0062622	90	2001: A Space Odyssey	1968	149	G	
14	tt0062622	90	2001: A Space Odyssey	1968	149	G	
15	tt0062622	90	2001: A Space Odyssey	1968	149	G	

### Query: Specific movie title, streaming service availability, price

Query in google\_movie\_streaming table for movie ("Inception"), streaming service available, and price

```

1 select google_movie_streaming.title, google_streaming_service.service_name, google_movie_streaming.price
2 from google_movie_streaming
3 join google_streaming_service
4 on google_movie_streaming.service_id = google_streaming_service.service_id
5 where title = 'Inception';

```

	title character varying	service_name character varying	price character varying
1	Inception	Amazon Prime Video	3.99
2	Inception	HBO Now	3.99
3	Inception	Philo	3.99
4	Inception	Showtime	Subscription
5	Inception	fuboTV	3.99
6	Inception	iTunes	3.99