12/19/2020

NetFlix Data

ETL Project report

1- Ahmad Abu Alafa
2- Hamid Zarringhalam
3- Loic Tiemani
4- Hazim Hamadneh

Contents

Introduction

In this project which is the continuation of Project-1, Netflix Data analysis. It is required to identify the source of Data sets, perform Extraction, Transformation and Loading (ETL) on these data sets.

ETL comprises of three methods:

1. Extraction
2. Transformation
3. Loading

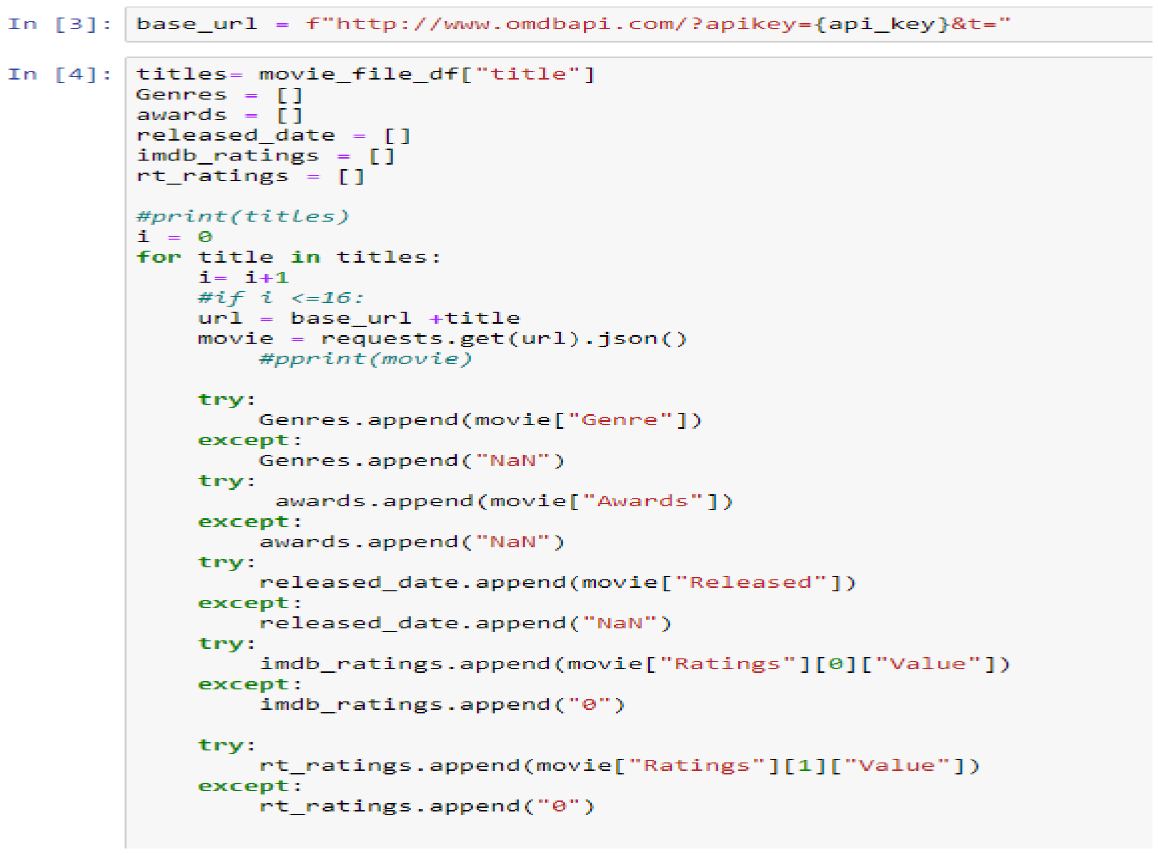
Extract Method

Data extraction includes the following tasks:

1. Extract relevant data from data sources:
    - https://www.kaggle.com/shivamb/netflix-shows?select=netflix_titles.csv
      The result of extracted data is in the form of CSV and it has the following fields:

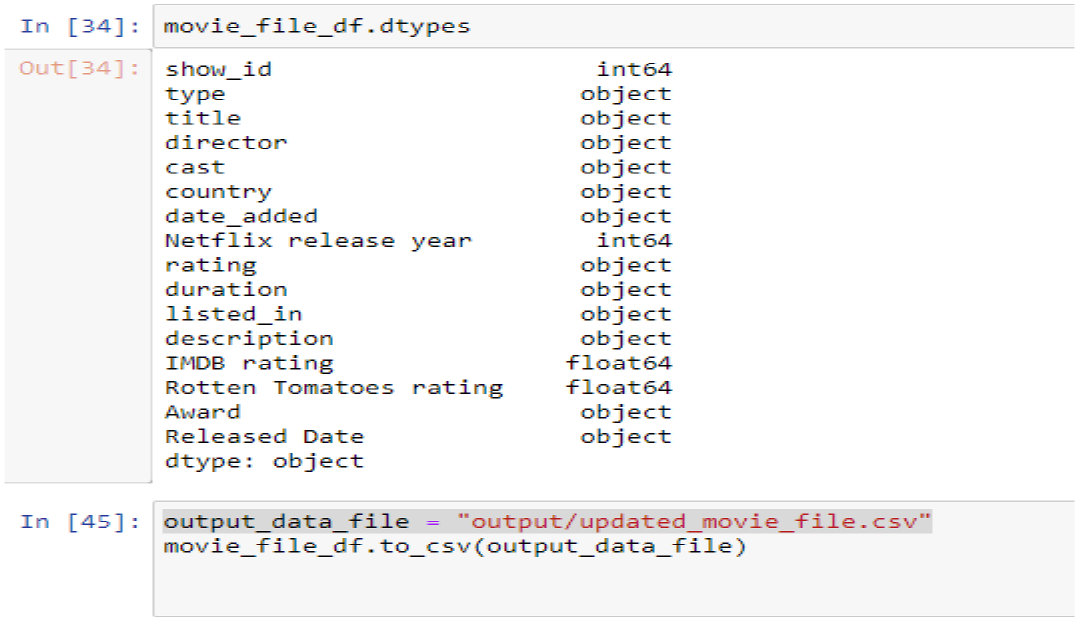
show_id	type	title	director	cast	country	date_adde	release_y	rating	duration	listed_in	descriptio

    - Public movie APIs:
        i. OMDB - OMDB to obtain information and metadata about movies.



```
In [3]: base_url = f"http://www.omdbapi.com/?apikey={api_key}&t="
```

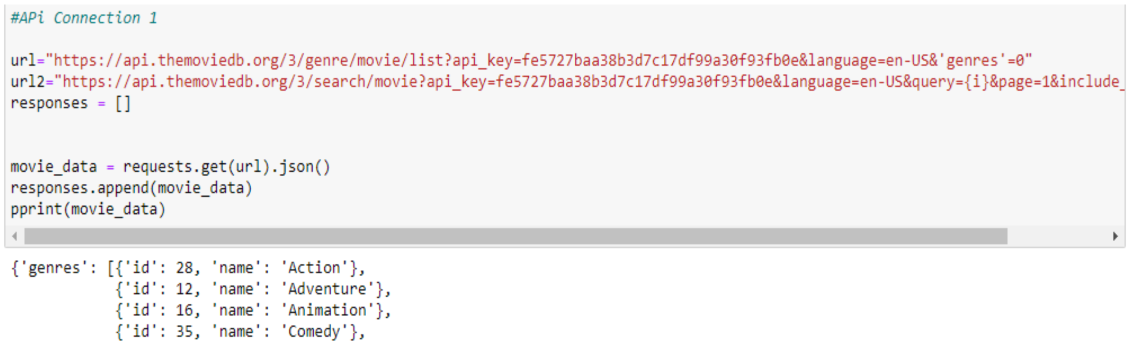
```
In [4]: titles= movie_file_df["title"]
        Genres = []
        awards = []
        released_date = []
        imdb_ratings = []
        rt_ratings = []

        #print(titles)
        i = 0
        for title in titles:
            i= i+1
            #if i <=16:
            url = base_url +title
            movie = requests.get(url).json()
                #pprint(movie)

            try:
                Genres.append(movie["Genre"])
            except:
                Genres.append("NaN")
            try:
                 awards.append(movie["Awards"])
            except:
                awards.append("NaN")
            try:
                released_date.append(movie["Released"])
            except:
                released_date.append("NaN")
            try:
                imdb_ratings.append(movie["Ratings"][0]["Value"])
            except:
                imdb_ratings.append("0")

            try:
                rt_ratings.append(movie["Ratings"][1]["Value"])
            except:
                rt_ratings.append("0")
```



```
In [34]: movie_file_df.dtypes
Out[34]: show_id                      int64
         type                        object
         title                       object
         director                    object
         cast                        object
         country                     object
         date_added                  object
         Netflix release year         int64
         rating                      object
         duration                    object
         listed_in                   object
         description                 object
         IMDB rating                float64
         Rotten Tomatoes rating     float64
         Award                       object
         Released Date               object
         dtype: object
```

```
In [45]: output_data_file = "output/updated_movie_file.csv"
         movie_file_df.to_csv(output_data_file)
```

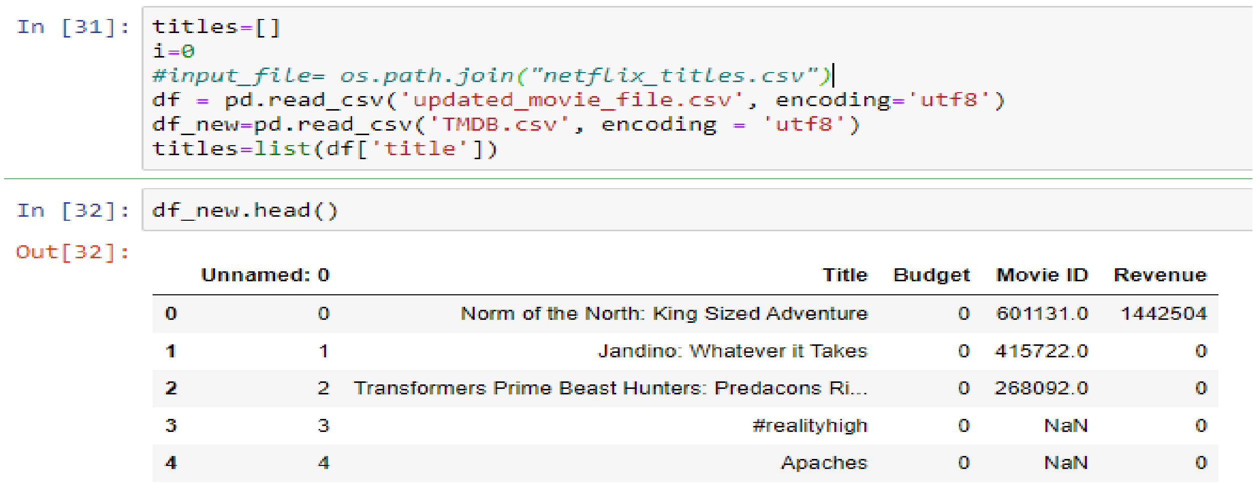
ii. TMDB - TMDB for movie discovery.



```
#APi Connection 1

url="https://api.themoviedb.org/3/genre/movie/list?api_key=fe5727baa38b3d7c17df99a30f93fb0e&language=en-US&'genres'=0"
url2="https://api.themoviedb.org/3/search/movie?api_key=fe5727baa38b3d7c17df99a30f93fb0e&language=en-US&query={i}&page=1&include_
responses = []


movie_data = requests.get(url).json()
responses.append(movie_data)
pprint(movie_data)
```

```
{'genres': [{'id': 28, 'name': 'Action'},
            {'id': 12, 'name': 'Adventure'},
            {'id': 16, 'name': 'Animation'},
            {'id': 35, 'name': 'Comedy'},
```

2. Reconcile records with the source data

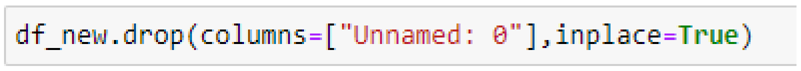


```
In [31]: titles=[]
         i=0
         #input_file= os.path.join("netflix_titles.csv")
         df = pd.read_csv('updated_movie_file.csv', encoding='utf8')
         df_new=pd.read_csv('TMDB.csv', encoding = 'utf8')
         titles=list(df['title'])
```

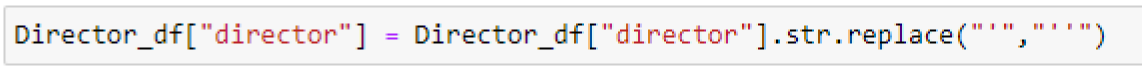
```
In [32]: df_new.head()
Out[32]:
```

	Unnamed: 0	Title	Budget	Movie ID	Revenue
0	0	Norm of the North: King Sized Adventure	0	601131.0	1442504
1	1	Jandino: Whatever it Takes	0	415722.0	0
2	2	Transformers Prime Beast Hunters: Predacons Ri...	0	268092.0	0
3	3	#realityhigh	0	NaN	0
4	4	Apaches	0	NaN	0

3. Make sure that no spam/unwanted data loaded

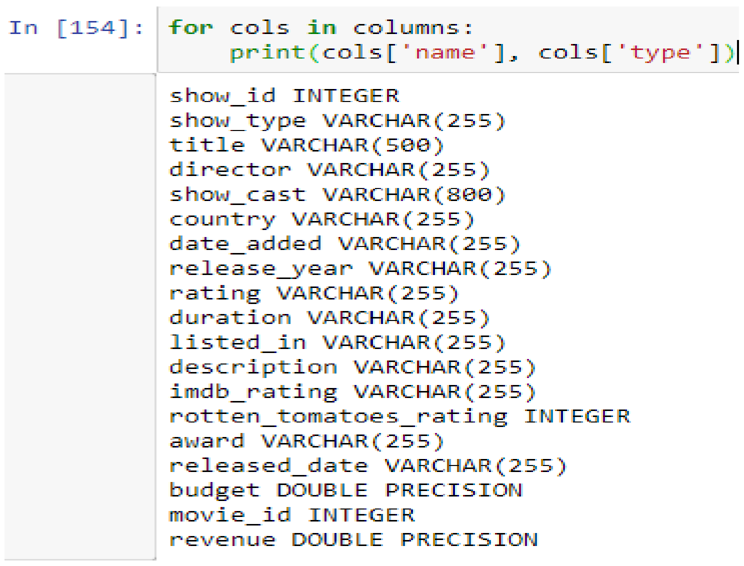


```
df_new.drop(columns=["Unnamed: 0"],inplace=True)
```



```
Director_df["director"] = Director_df["director"].str.replace("'","''")
```

4. Data type check



```
In [154]: for cols in columns:
              print(cols['name'], cols['type'])

          show_id INTEGER
          show_type VARCHAR(255)
          title VARCHAR(500)
          director VARCHAR(255)
          show_cast VARCHAR(800)
          country VARCHAR(255)
          date_added VARCHAR(255)
          release_year VARCHAR(255)
          rating VARCHAR(255)
          duration VARCHAR(255)
          listed_in VARCHAR(255)
          description VARCHAR(255)
          imdb_rating VARCHAR(255)
          rotten_tomatoes_rating INTEGER
          award VARCHAR(255)
          released_date VARCHAR(255)
          budget DOUBLE PRECISION
          movie_id INTEGER
          revenue DOUBLE PRECISION
```

Data types of all the fields were checked and identified as below:

NetFlix CSV output data

Field Name:	**Description:**	**Example:**	**Data Type:**	**Unique:**	**Null:**	**Comments:**
show_id	Unique ID for every Movie / Tv Show	81145628	String	Yes	No	
type	Identifier - A Movie or TV Show	Movie	String	No	No	
title	Title of the Movie / Tv Show	Norm of the North: King Sized Adventure	String	No	No	
director	Director of the Movie	Richard Finn, Tim Maltby	String	No	yes	Multiple names in each field
cast	Actors involved in the movie / show	Alan Marriott, Andrew Toth, Brian Dobson, Cole Howard, Jennifer Cameron, Jonathan Holmes, Lee Tockar...	String	No	Yes	Multiple names in each field
country	Country where the movie / show was produced	United States, India, South Korea, China	String	No	yes	Multiple names in each field
date_added	Date it was added on Netflix	September 9, 2019	Date	No	yes	
release_year	Actual Release year of the move / show text_formatratingsort TV Rating of the movie /	2019	Integer	No	No	

rating	TV Rating of the movie / show	TV-PG	String	No	Yes	
duration	Total Duration - in minutes or number of seasons	90 min	String	No	No	
listed_in	Genre	Children & Family Movies, Comedies	String	No	No	Multiple names in each field
description	The summary description	Before planning an awesome wedding for his grandfather, a polar bear king must	String	No	No	

This is OMDB API extract:

Field Name:	Description:	Example:	Data Type:	Uniqu e:	Null:	Comments:
Title	Title of the Movie / Tv Show	Man of Steel	String	No	No	
Year	Actual Release year of the move / show text_formatratingsort TV Rating of the movie /	2013	Intege r	No	No	
Rated	TV Rating of the movie / show	PG-13	String	No	Yes	
Released	Release date	14 Jun 2013	String	No	Yes	
Runtime	Duration	143 min	String	No	Yes	
Genre	Type of movie	Action,Adventur e, Sci-Fi	String	No	No	
Director	Director	Zack Snyder	String	No	No	
Writer	Author	David S. Goyer (screenplay)	String	No	No	

		David S. Goyer (story)				
Actors	Actor and Actress	Henry Cavill Amy Adams Michael Shannon Diane Lane	String	No	No	
Plot	Description	An alien child is evacuated from his dying world and sent to Earth to live among humans. His peace is threatened when other survivors of his home planet invade Earth	String	Yes	No	
Language		English	String	No	No	
Country	Country of producing the product	UA, UK	String	No	No	
Awards	Received Awards for the product	7 wins & 46 nominations	String	No	Yes	
Poster	create by the studio used to promote or advertise the title	N/A	String	No	No	
Source	IMDb registered users who casts a vote	N/A	String	No	Yes	
Value	Accurate Percentage of the Rating of respected critics	56%	Float	No	Yes	
Source	Rating of critics	Rotten Tomatoes	String	No	Yes	
Value	Percentage of the Rating of respected critics	55/100	Float	No	Yes	
Metascore	Rating of respected critics	55	Int	No	Yes	
imdbRating	Rating on IMDB	7.0	Int	No	Yes	

imdbVotes	Number of votes for the product	92	Int	No	Yes	
imdbID	IMDB Movie ID Number	tt0770828	String	Yes	No	
Type	Movie or TV-Series	movie	Float	No	No	
DVD	If the product is available on DVD	N/A	String	No	Yes	
BoxOffice	Box Office receives data from a variety of sources, including film studios, distributors, and production companies from around the world	N/A	String	No	Yes	
Production	Production company	Syncopy	String	No	Yes	

This is TMDB extracted fields

Field Name:	Description:	Example	Data Type	Unique	Null	Comments
adult: TRUE/FALSE	Restricted for adult	Yes	Boolean	No	Yes	
backdrop_path: varchar(255)			String			
genre_ids: Numbers (series)	The type of movie, Comeddy,…	Action,Adventure, Sci-Fi	String	No	No	
id: varchar (10)	The Movie or Show ID	415722	String	Yes	No	
original_language: char (2)	Original Language that the movie is produced	en	String	No	no	
original_title: varchar (255)	The original title that is chosen	Norm of the North: King Sized Adventure	String	No	No	

overview: varchar (1500)	General view of summery	An ancient Chinese artifact has been stolen by a villainous archaeologist named Dexter. With the help of his lemming friends, Norm must keep his word and embark on a journey across the world to help recover the artifact for the people of China.	String	Yes	Yes	
popularity: float	Popularity based on number of votes	10	Float	No	Yes	
poster_path: varchar(255)	create by the studio used to promote or advertise the title	N/A	String	No	No	
release_date: date	The Release date of the product	14 June 2013	String	no	yes	
title: varchar (255)	The title of the movie ot TV-Series	Norm of the North: King Sized Adventure	String	No	No	
video: TRUE/FALSE	On Video tape or not	True	Boolean	No	Yes	
vote average: float	Min Average of Votes for the movie	10	Float	No	Yes	
vote_count: int (edited)	Number of Votes	10	Int	No	Yes	

Transform Method:

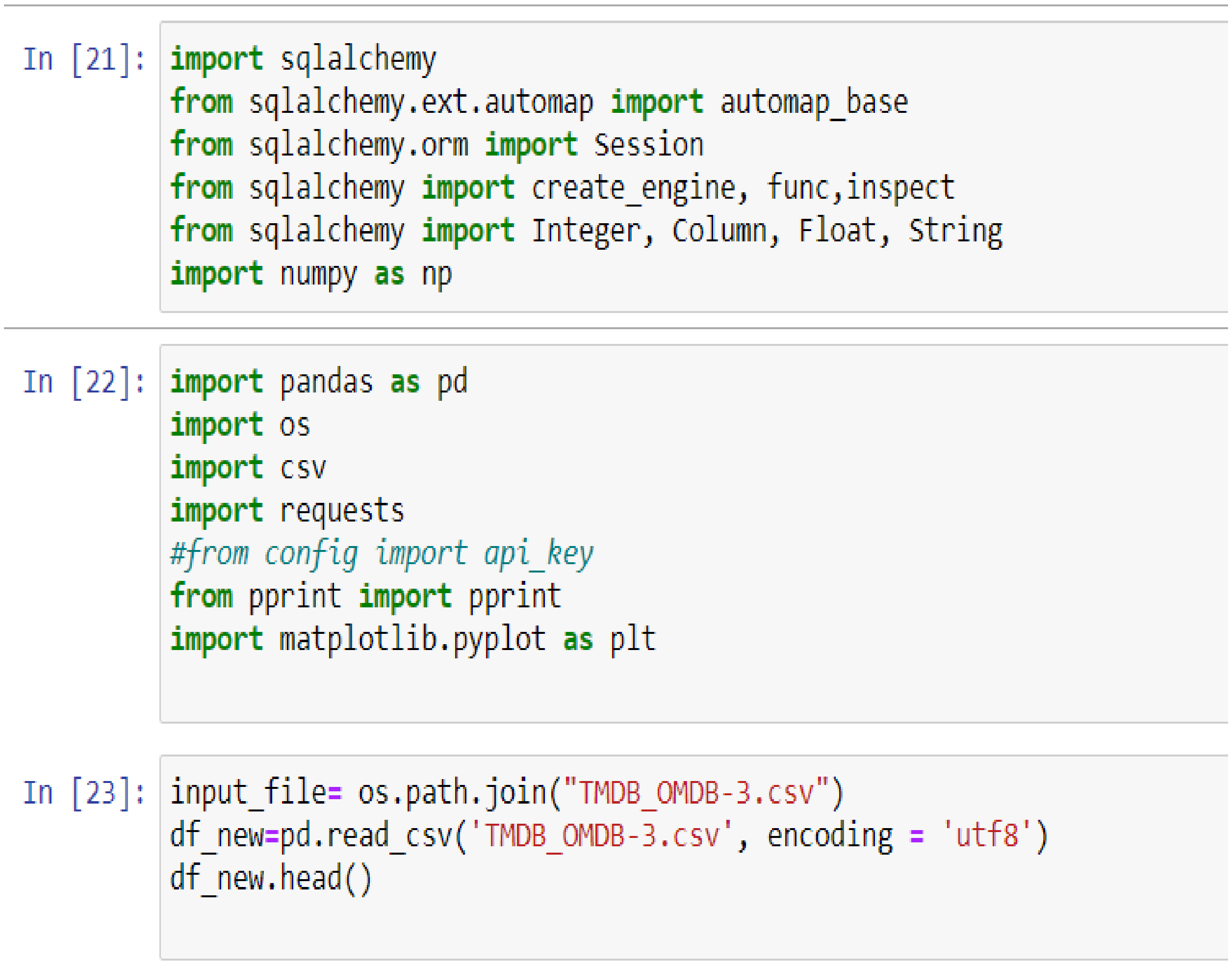
Data transformation includes the following tasks:

1. Data Aggregation of different sources to one:

Both API outputs and Netflix CSV merged into one CSV which conclude the following fields:

show_id	type	title	director	cast	country	date_adde	Netflix re	rating	duration	listed_in

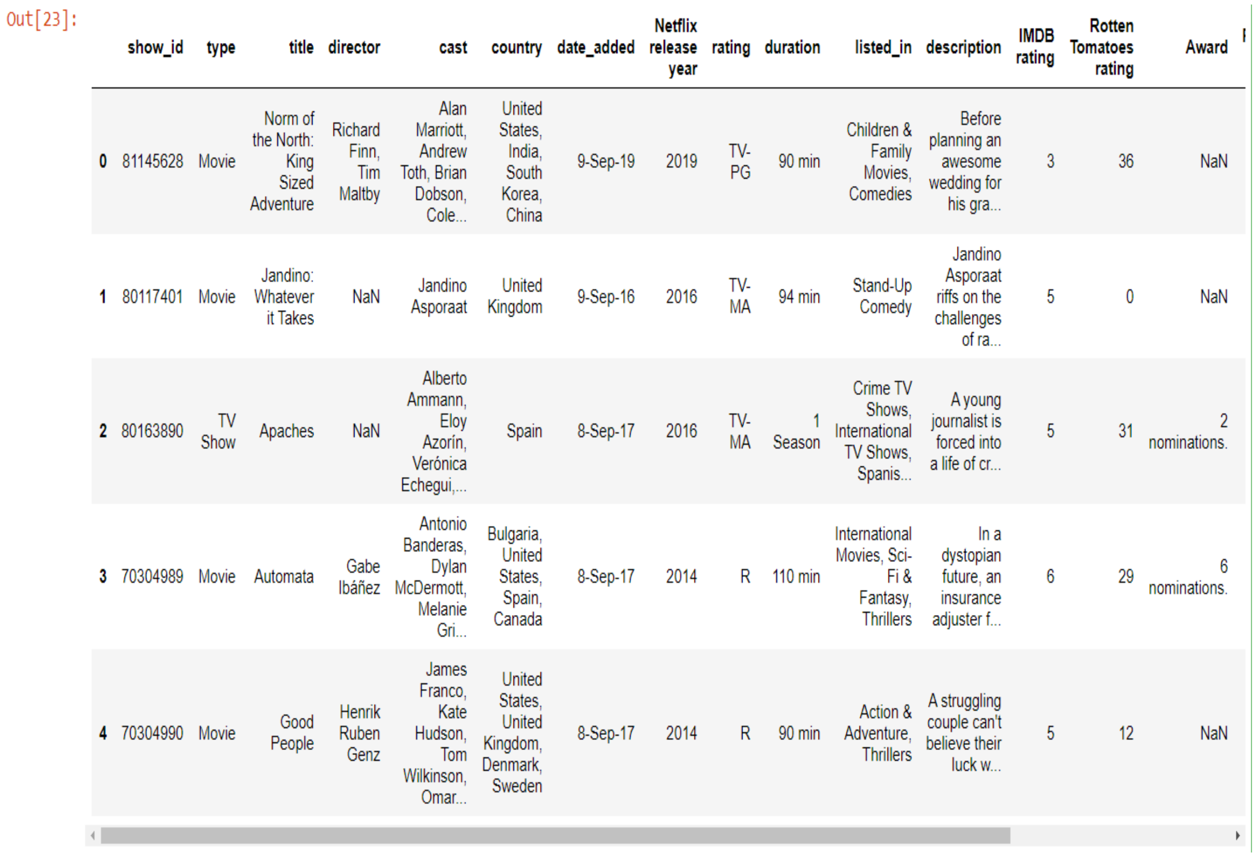
descriptio	IMDB ratir	Rotten To	Award	Released	IMDB ratir	Title	Budget	Movie ID	Revenue



In [21]:
```
import sqlalchemy
from sqlalchemy.ext.automap import automap_base
from sqlalchemy.orm import Session
from sqlalchemy import create_engine, func,inspect
from sqlalchemy import Integer, Column, Float, String
import numpy as np
```

In [22]:
```
import pandas as pd
import os
import csv
import requests
#from config import api_key
from pprint import pprint
import matplotlib.pyplot as plt
```

In [23]:
```
input_file= os.path.join("TMDB_OMDB-3.csv")
df_new=pd.read_csv('TMDB_OMDB-3.csv', encoding = 'utf8')
df_new.head()
```



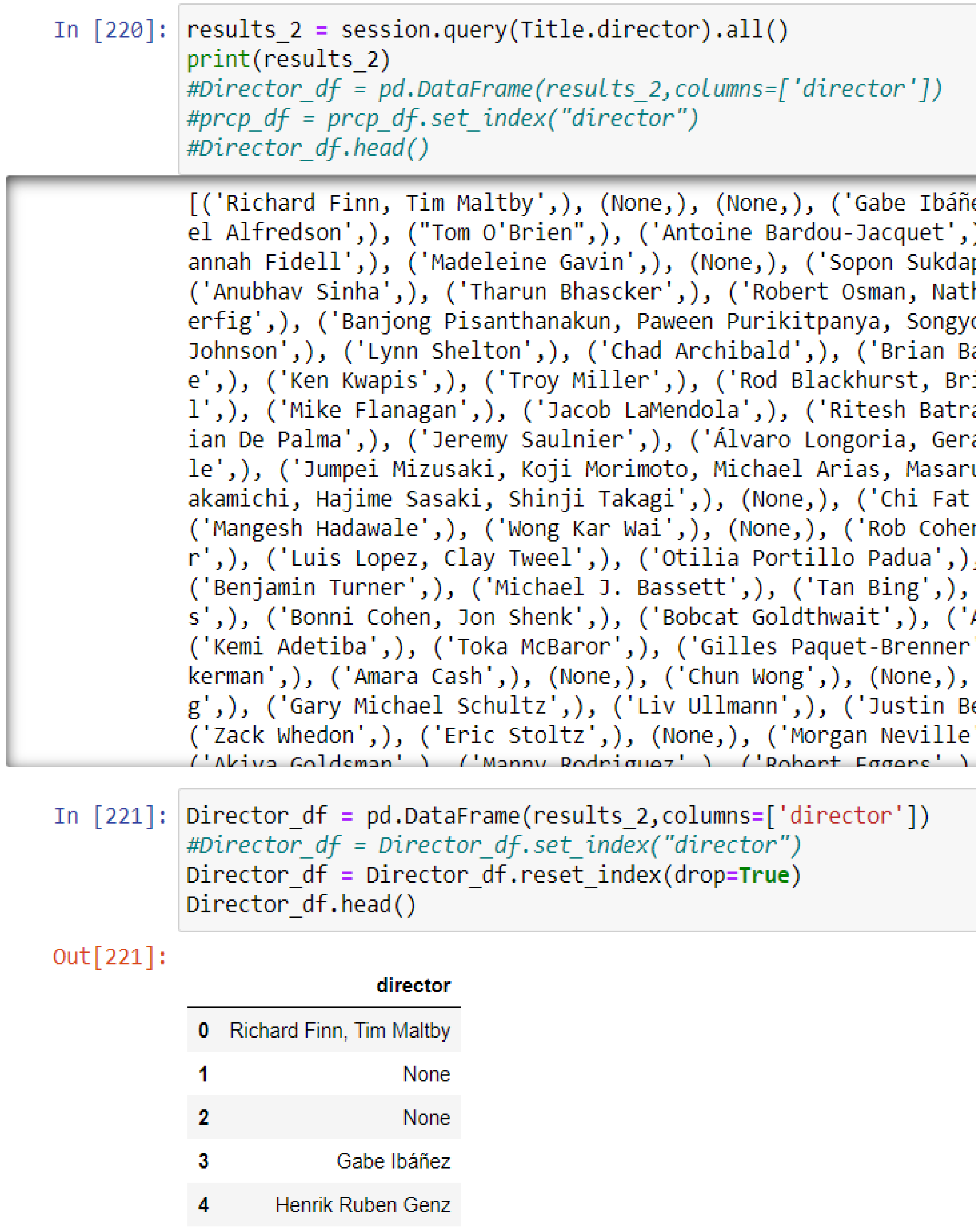
Out[23]:

	show_id	type	title	director	cast	country	date_added	Netflix release year	rating	duration	listed_in	description	IMDB rating	Rotten Tomatoes rating	Award
0	81145628	Movie	Norm of the North: King Sized Adventure	Richard Finn, Tim Maltby	Alan Marriott, Andrew Toth, Brian Dobson, Cole...	United States, India, South Korea, China	9-Sep-19	2019	TV-PG	90 min	Children & Family Movies, Comedies	Before planning an awesome wedding for his gra...	3	36	NaN
1	80117401	Movie	Jandino: Whatever it Takes	NaN	Jandino Asporaat	United Kingdom	9-Sep-16	2016	TV-MA	94 min	Stand-Up Comedy	Jandino Asporaat riffs on the challenges of ra...	5	0	NaN
2	80163890	TV Show	Apaches	NaN	Alberto Ammann, Eloy Azorín, Verónica Echegui,...	Spain	8-Sep-17	2016	TV-MA	1 Season	Crime TV Shows, International TV Shows, Spanis...	A young journalist is forced into a life of cr...	5	31	2 nominations.
3	70304989	Movie	Automata	Gabe Ibáñez	Antonio Banderas, Dylan McDermott, Melanie Gri...	Bulgaria, United States, Spain, Canada	8-Sep-17	2014	R	110 min	International Movies, Sci-Fi & Fantasy, Thrillers	In a dystopian future, an insurance adjuster f...	6	29	6 nominations.
4	70304990	Movie	Good People	Henrik Ruben Genz	James Franco, Kate Hudson, Tom Wilkinson, Omar...	United States, United Kingdom, Denmark, Sweden	8-Sep-17	2014	R	90 min	Action & Adventure, Thrillers	A struggling couple can't believe their luck w...	5	12	NaN

Data Cleaning

There were some fields that are duplicate, others are no value and others have more than one information, such as Director, Cast, Country or listed in.

Example:



```
In [220]: results_2 = session.query(Title.director).all()
          print(results_2)
          #Director_df = pd.DataFrame(results_2,columns=['director'])
          #prcp_df = prcp_df.set_index("director")
          #Director_df.head()
```

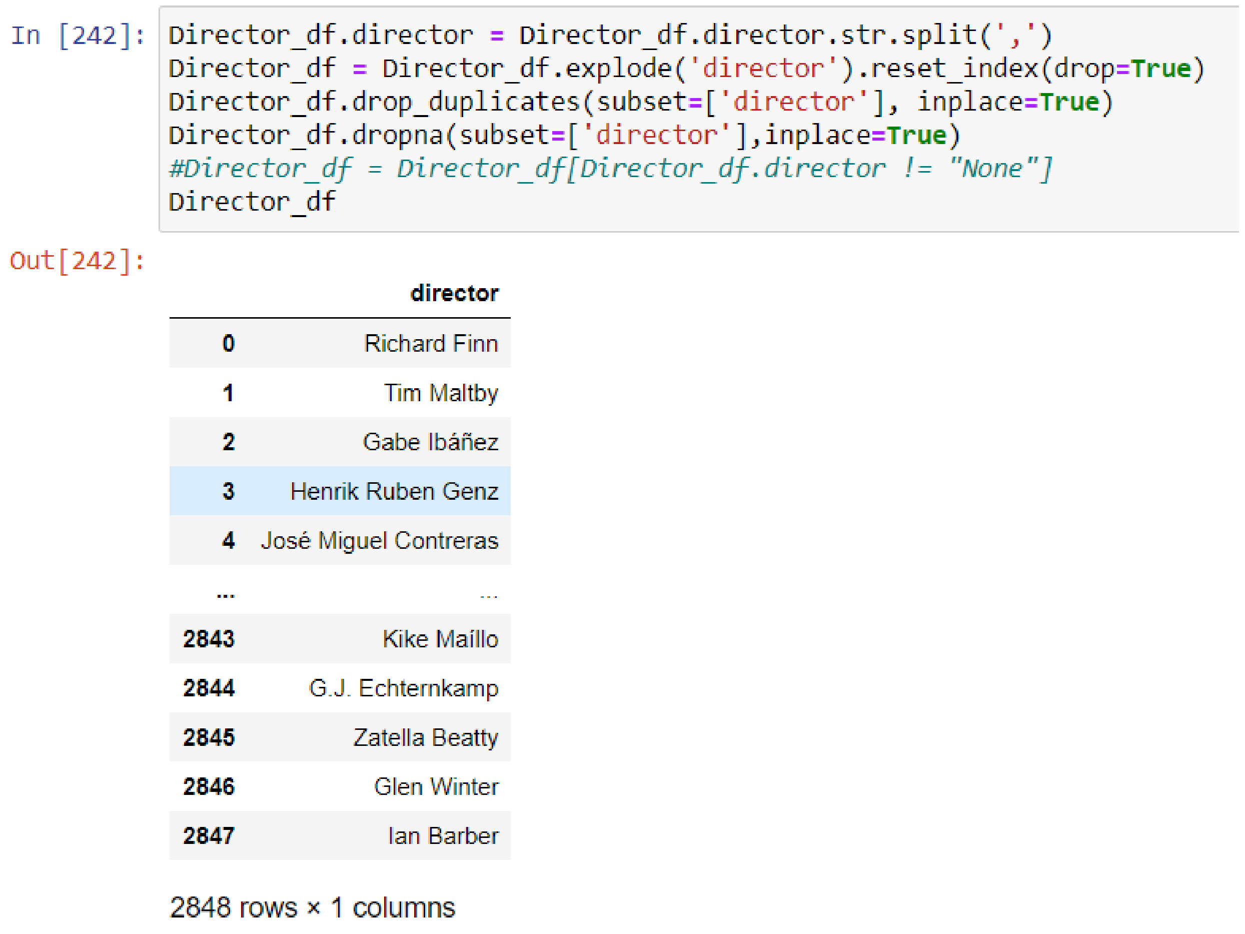
```
[('Richard Finn, Tim Maltby',), (None,), (None,), ('Gabe Ibáñe
el Alfredson',), ("Tom O'Brien",), ('Antoine Bardou-Jacquet',)
annah Fidell',), ('Madeleine Gavin',), (None,), ('Sopon Sukdap
('Anubhav Sinha',), ('Tharun Bhascker',), ('Robert Osman, Nath
erfig',), ('Banjong Pisanthanakun, Paween Purikitpanya, Songyo
Johnson',), ('Lynn Shelton',), ('Chad Archibald',), ('Brian Ba
e',), ('Ken Kwapis',), ('Troy Miller',), ('Rod Blackhurst, Bri
l',), ('Mike Flanagan',), ('Jacob LaMendola',), ('Ritesh Batra
ian De Palma',), ('Jeremy Saulnier',), ('Álvaro Longoria, Gera
le',), ('Jumpei Mizusaki, Koji Morimoto, Michael Arias, Masaru
akamichi, Hajime Sasaki, Shinji Takagi',), (None,), ('Chi Fat
('Mangesh Hadawale',), ('Wong Kar Wai',), (None,), ('Rob Coher
r',), ('Luis Lopez, Clay Tweel',), ('Otilia Portillo Padua',),
('Benjamin Turner',), ('Michael J. Bassett',), ('Tan Bing',),
s',), ('Bonni Cohen, Jon Shenk',), ('Bobcat Goldthwait',), ('A
('Kemi Adetiba',), ('Toka McBaror',), ('Gilles Paquet-Brenner'
kerman',), ('Amara Cash',), (None,), ('Chun Wong',), (None,),
g',), ('Gary Michael Schultz',), ('Liv Ullmann',), ('Justin Be
('Zack Whedon',), ('Eric Stoltz',), (None,), ('Morgan Neville'
('Akiva Goldsman',), ('Manny Rodriguez',), ('Robert Eggers',)
```

```
In [221]: Director_df = pd.DataFrame(results_2,columns=['director'])
          #Director_df = Director_df.set_index("director")
          Director_df = Director_df.reset_index(drop=True)
          Director_df.head()
```

Out[221]:

	director
0	Richard Finn, Tim Maltby
1	None
2	None
3	Gabe Ibáñez
4	Henrik Ruben Genz

In this process, we removed the duplicates, NaN and configured the proper indexing as well.



```
In [242]: Director_df.director = Director_df.director.str.split(',')
          Director_df = Director_df.explode('director').reset_index(drop=True)
          Director_df.drop_duplicates(subset=['director'], inplace=True)
          Director_df.dropna(subset=['director'],inplace=True)
          #Director_df = Director_df[Director_df.director != "None"]
          Director_df
```

```
Out[242]:
                     director
             0           Richard Finn
             1             Tim Maltby
             2            Gabe Ibáñez
             3       Henrik Ruben Genz
             4  José Miguel Contreras
           ...                    ...
          2843            Kike Maillo
          2844        G.J. Echternkamp
          2845          Zatella Beatty
          2846             Glen Winter
          2847              Ian Barber

          2848 rows × 1 columns
```

OMDB-TMDB Table which is consolidated of the data from Netflix CSV, TMDB API and OMDB API

Field Name:	**Description:**	**Example**	**Data Type**	**Unique**	**Null**	**Comments**
show ID	ID number of the movie	81145628	Int	Yes	No	
Show type	Type of product: Movie/TV Series	Movie	String	Yes	No	
Title	Title of the Movie / Tv Show	Man of Steel	String	No	No	
Date Added	Date added	8-sep-2017	String	No	Yes	

Netflix release year	Release date	14 Jun 2013	String	No	Yes	
rating	TV Rating of the movie / show	TV-PG	String	No	Yes	
director	Director	Zack Snyder	String	No	No	
cast	Actors involved in the movie / show	Alan Marriott, Andrew Toth, Brian Dobson, Cole Howard, Jennifer Cameron, Jonathan Holmes, Lee Tockar...	String	No	Yes	Multiple names in each field
duration	Total Duration - in minutes or number of seasons	90 min	String	No	No	
listed_in	Genre	Children & Family Movies, Comedies	String	No	No	Multiple names in each field
description	The summary description	Before planning an awesome wedding for his grandfather, a polar bear king must	String	No	No	
IMDB rating	Rating on IMDB	7.0	Int	No	Yes	
Rotten Tomatoes rating	Rating of critics	Rotten Tomatoes	String	No	Yes	

Award	Received Awards for the product	7 wins & 46 nominations	String	No	Yes	
Released Date	Release date	14 Jun 2013	String	No	Yes	
Budget	The budget for the product	150,000,000	Int	No	Yes	
Movie ID	The OMDB ID for the movie	415722	Int	yes	yes	
Revenue	The income of the product	750,000,000	Int	No	Yes	

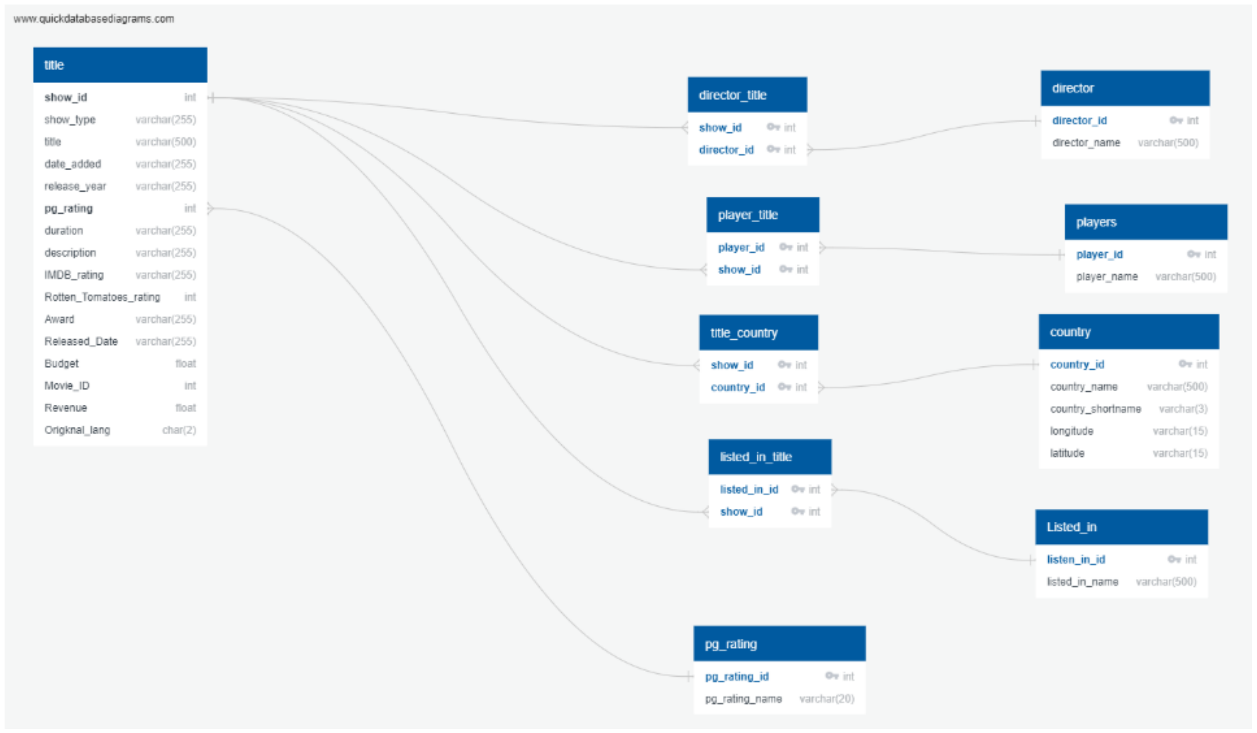
Database tables designed

Main table is representing the show titles and its related properties.
This table has been normalized to first normal form by separating entities in different tables and relations have been created to connect each entity to its related data. For example: Titles and Players, Titles and Directors, etc.

Necessary tables and the relations between them was designed.

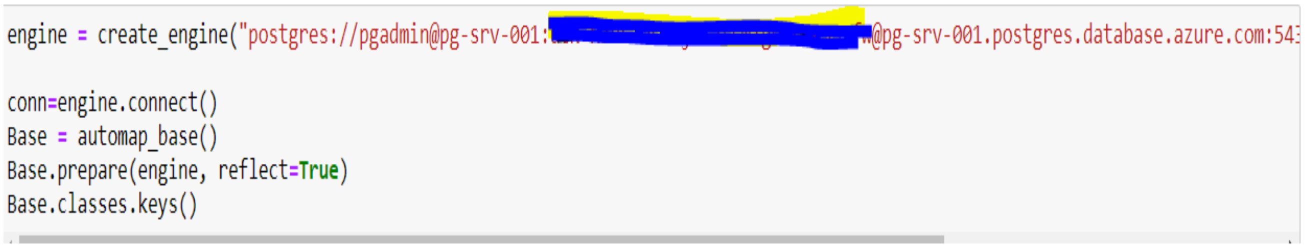
The following were considered to separate the related data from each other and have proper connections between the tables.

ERD Diagram:



SQL Server identified and connected

Azure cloud PostgreSQL standalone server selected and used



```
engine = create_engine("postgres://pgadmin@pg-srv-001:[redacted]@pg-srv-001.postgres.database.azure.com:54

conn=engine.connect()
Base = automap_base()
Base.prepare(engine, reflect=True)
Base.classes.keys()
```

2. Tables created on the server

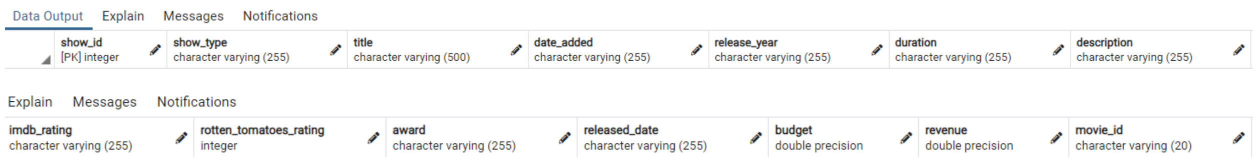
TABLE SCHEMA:

```
CREATE TABLE "title" (
    "show_id" int   NOT NULL,
    "show_type" varchar(255)   NOT NULL,
    "title" varchar(500)   NOT NULL,
    "date_added" varchar(255)   NOT NULL,
    "release_year" varchar(255)   NOT NULL,
    "pg_rating" int   NOT NULL,
    "duration" varchar(255)   NOT NULL,
    "description" varchar(255)   NOT NULL,
    "IMDB_rating" varchar(255)   NOT NULL,
    "Rotten_Tomatoes_rating" int   NOT NULL,
    "Award" varchar(255)   NOT NULL,
    "Released_Date" varchar(255)   NOT NULL,
    "Budget" float   NOT NULL,
    "Movie_ID" int   NOT NULL,
    "Revenue" float   NOT NULL,
    "Origknal_lang" char(2)   NOT NULL
);

CREATE TABLE "title_country" (
    "show_id" int   NOT NULL,
    "country_id" int   NOT NULL,
    CONSTRAINT "pk_title_country" PRIMARY KEY (
        "show_id","country_id"
     )
);

CREATE TABLE "country" (
    "country_id" int   NOT NULL,
    "country_name" varchar(500)   NOT NULL,
    "country_shortname" varchar(3)   NOT NULL,
    "longitude" varchar(15)   NOT NULL,
    "latitude" varchar(15)   NOT NULL,
    CONSTRAINT "pk_country" PRIMARY KEY (
        "country_id"
     )
);

CREATE TABLE "Listed_in" (
    "listen_in_id" int   NOT NULL,
    "listed_in_name" varchar(500)   NOT NULL,
    CONSTRAINT "pk_Listed_in" PRIMARY KEY (
        "listen_in_id"
     )
);
```

```
CREATE TABLE "listed_in_title" (
    "listed_in_id" int   NOT NULL,
    "show_id" int   NOT NULL,
    CONSTRAINT "pk_listed_in_title" PRIMARY KEY (
        "listed_in_id","show_id"
     )
);

CREATE TABLE "director" (
    "director_id" int   NOT NULL,
    "director_name" varchar(500)   NOT NULL,
    CONSTRAINT "pk_director" PRIMARY KEY (
        "director_id"
     )
);

CREATE TABLE "director_title" (
    "show_id" int   NOT NULL,
    "director_id" int   NOT NULL,
    CONSTRAINT "pk_director_title" PRIMARY KEY (
        "show_id","director_id"
     )
);

CREATE TABLE "players" (
    "player_id" int   NOT NULL,
    "player_name" varchar(500)   NOT NULL,
    CONSTRAINT "pk_players" PRIMARY KEY (
        "player_id"
     )
);

CREATE TABLE "player_title" (
    "player_id" int   NOT NULL,
    "show_id" int   NOT NULL,
    CONSTRAINT "pk_player_title" PRIMARY KEY (
        "player_id","show_id"
     )
);

CREATE TABLE "pg_rating" (
    "pg_rating_id" int   NOT NULL,
    "pg_rating_name" varchar(20)   NOT NULL,
    CONSTRAINT "pk_pg_rating" PRIMARY KEY (
        "pg_rating_id"
     )
);

ALTER TABLE "title" ADD CONSTRAINT "fk_title_pg_rating" FOREIGN
KEY("pg_rating")
REFERENCES "pg_rating" ("pg_rating_id");
```

```
ALTER TABLE "title_country" ADD CONSTRAINT "fk_title_country_show_id"
FOREIGN KEY("show_id")
REFERENCES "title" ("show_id");

ALTER TABLE "title_country" ADD CONSTRAINT
"fk_title_country_country_id" FOREIGN KEY("country_id")
REFERENCES "country" ("country_id");

ALTER TABLE "listed_in_title" ADD CONSTRAINT
"fk_listed_in_title_listed_in_id" FOREIGN KEY("listed_in_id")
REFERENCES "Listed_in" ("listen_in_id");

ALTER TABLE "listed_in_title" ADD CONSTRAINT
"fk_listed_in_title_show_id" FOREIGN KEY("show_id")
REFERENCES "title" ("show_id");

ALTER TABLE "director_title" ADD CONSTRAINT
"fk_director_title_show_id" FOREIGN KEY("show_id")
REFERENCES "title" ("show_id");

ALTER TABLE "director_title" ADD CONSTRAINT
"fk_director_title_director_id" FOREIGN KEY("director_id")
REFERENCES "director" ("director_id");

ALTER TABLE "player_title" ADD CONSTRAINT "fk_player_title_player_id"
FOREIGN KEY("player_id")
REFERENCES "players" ("player_id");

ALTER TABLE "player_title" ADD CONSTRAINT "fk_player_title_show_id"
FOREIGN KEY("show_id")
REFERENCES "title" ("show_id");
```

<u>"Title" Table:</u>



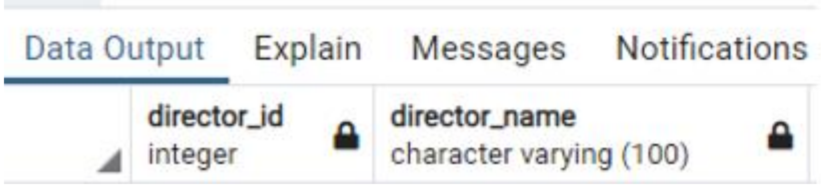
Data Output Explain Messages Notifications

show_id	show_type	title	date_added	release_year	duration	description
[PK] integer	character varying (255)	character varying (500)	character varying (255)	character varying (255)	character varying (255)	character varying (255)

Explain Messages Notifications

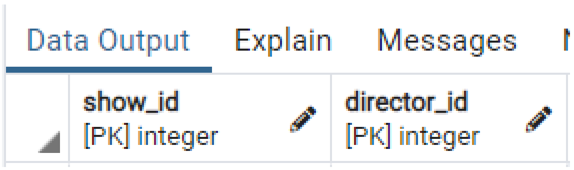
imdb_rating	rotten_tomatoes_rating	award	released_date	budget	revenue	movie_id
character varying (255)	integer	character varying (255)	character varying (255)	double precision	double precision	character varying (20)

"Director" Table:



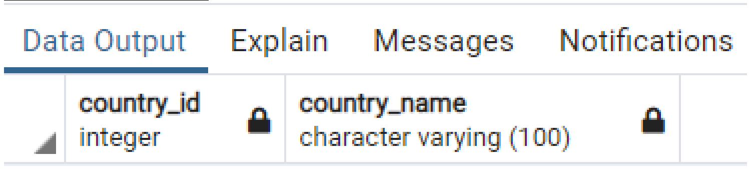
director_id integer	director_name character varying (100)

"Director_title" Table:



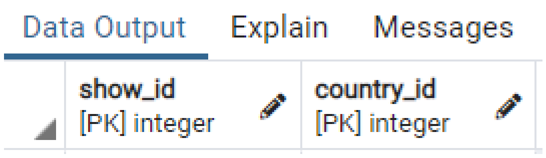
show_id [PK] integer	director_id [PK] integer

"Country "Table:



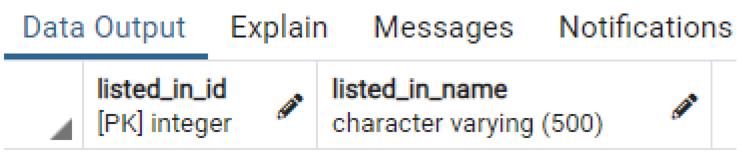
country_id integer	country_name character varying (100)

"Title_country" Table:



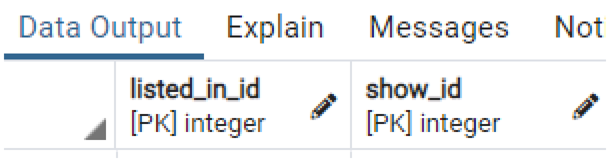
show_id [PK] integer	country_id [PK] integer

"Listed_in" Table:



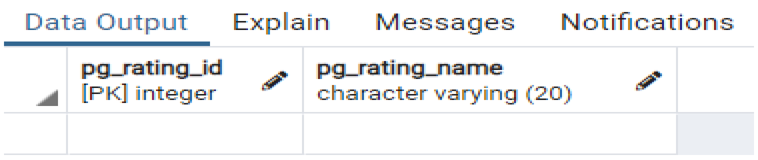
listed_in_id [PK] integer	listed_in_name character varying (500)

"Listed_in_title" Table:



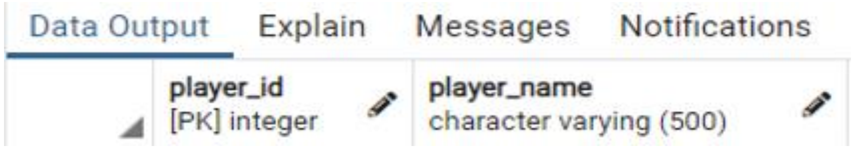
listed_in_id [PK] integer	show_id [PK] integer

<u>"PG_Rating" Table:</u>



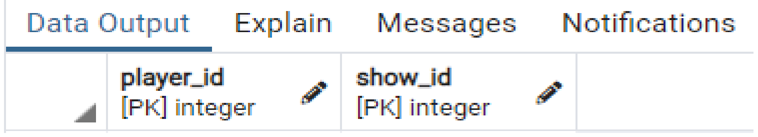
pg_rating_id [PK] integer	pg_rating_name character varying (20)

<u>"Players" Table:</u>



player_id [PK] integer	player_name character varying (500)

<u>"Players_Title" Table:</u>

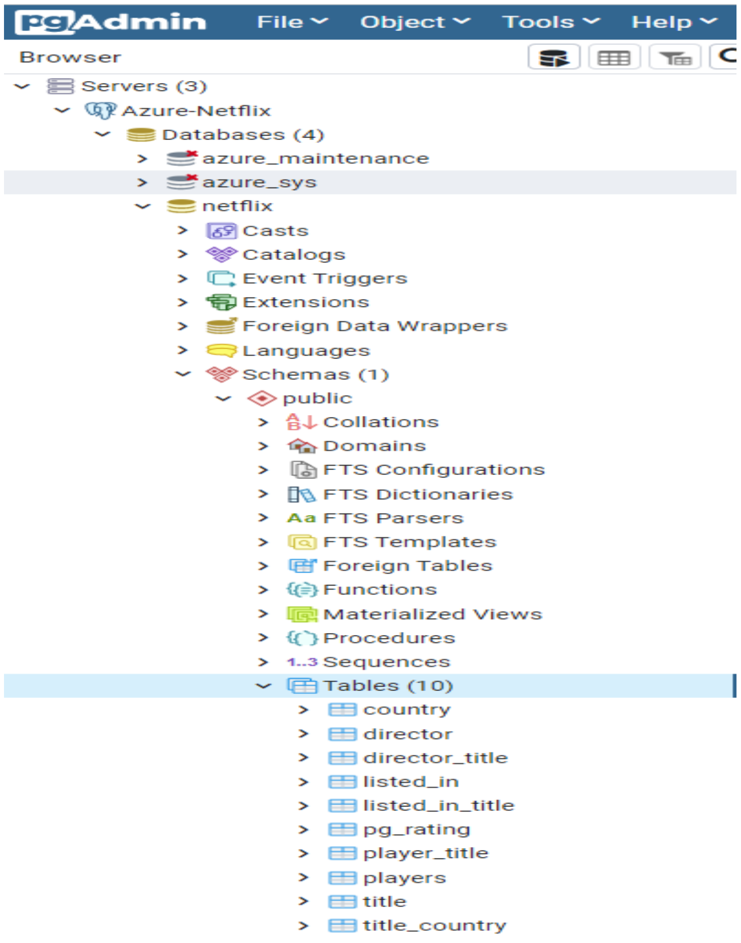


player_id [PK] integer	show_id [PK] integer

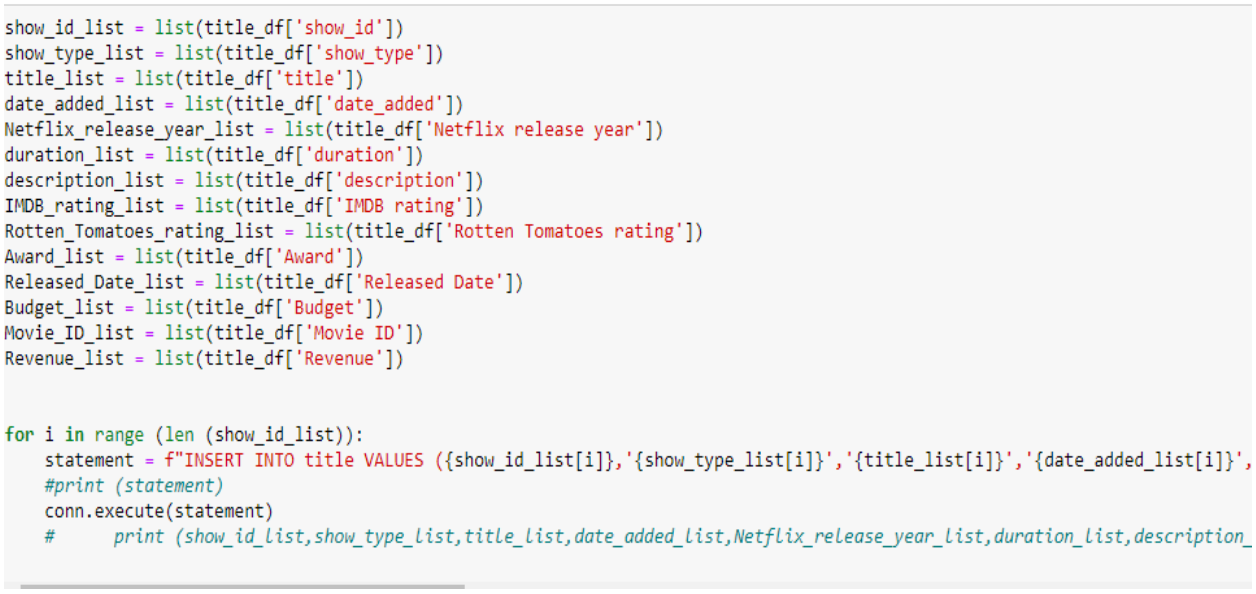
And as a result, we have multiple tables on the server as below:

Loading Method

After creating the tables, we imported the data to the tables based on the build design of the tables. Example of loading data into Title table:

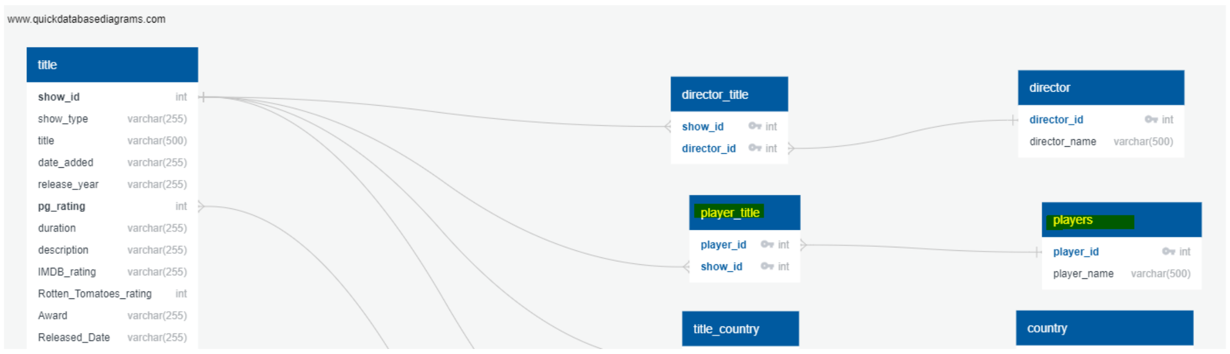


```
Servers (3)
  Azure-Netflix
    Databases (4)
      azure_maintenance
      azure_sys
      netflix
        Casts
        Catalogs
        Event Triggers
        Extensions
        Foreign Data Wrappers
        Languages
        Schemas (1)
          public
            Collations
            Domains
            FTS Configurations
            FTS Dictionaries
            FTS Parsers
            FTS Templates
            Foreign Tables
            Functions
            Materialized Views
            Procedures
            Sequences
            Tables (10)
              country
              director
              director_title
              listed_in
              listed_in_title
              pg_rating
              player_title
              players
              title
              title_country
```



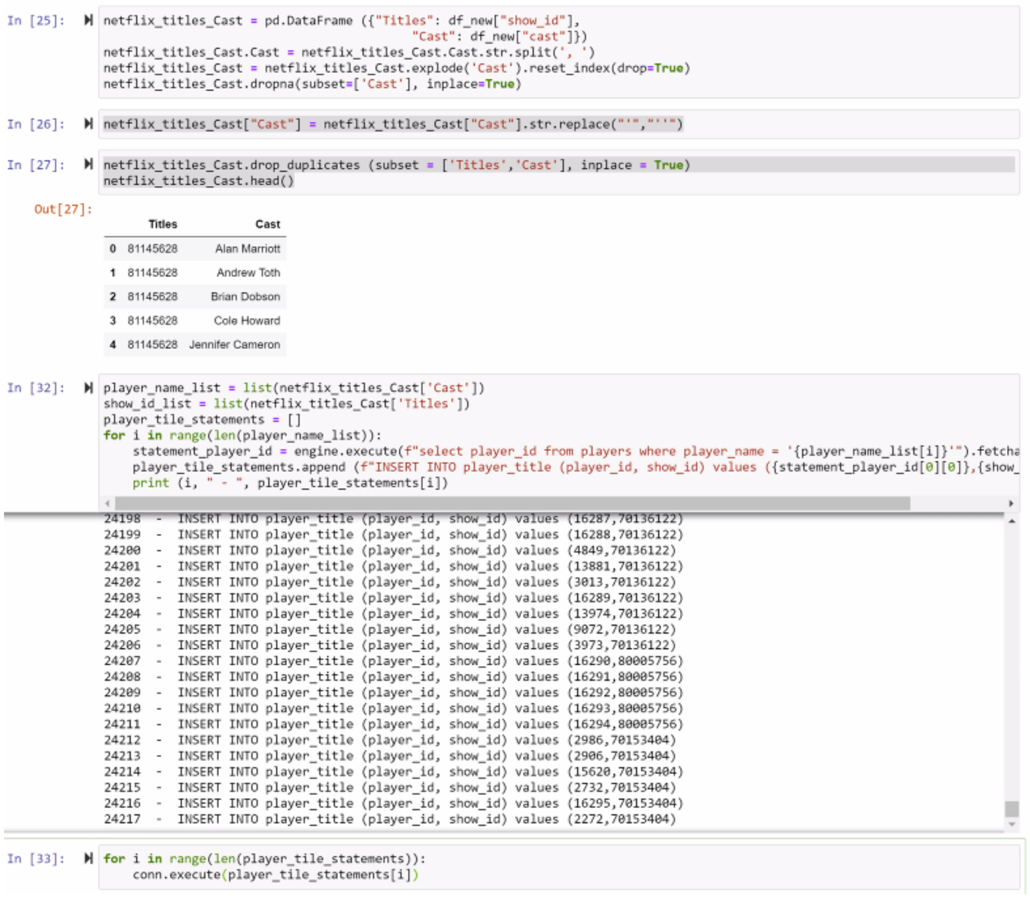
```
show_id_list = list(title_df['show_id'])
show_type_list = list(title_df['show_type'])
title_list = list(title_df['title'])
date_added_list = list(title_df['date_added'])
Netflix_release_year_list = list(title_df['Netflix release year'])
duration_list = list(title_df['duration'])
description_list = list(title_df['description'])
IMDB_rating_list = list(title_df['IMDB rating'])
Rotten_Tomatoes_rating_list = list(title_df['Rotten Tomatoes rating'])
Award_list = list(title_df['Award'])
Released_Date_list = list(title_df['Released Date'])
Budget_list = list(title_df['Budget'])
Movie_ID_list = list(title_df['Movie ID'])
Revenue_list = list(title_df['Revenue'])


for i in range (len (show_id_list)):
    statement = f"INSERT INTO title VALUES ({show_id_list[i]},'{show_type_list[i]}','{title_list[i]}','{date_added_list[i]}',
    #print (statement)
    conn.execute(statement)
    #     print (show_id_list,show_type_list,title_list,date_added_list,Netflix_release_year_list,duration_list,description_
```

For tables that have Join table such as “Players” which has “Player_title” as its join table. before importing the data to join table, we needed data imported to “Players” table first:



The table “player_title” is used to resolve the M-M relationship.

Example of loading data from “Players” table to “Player_title” table

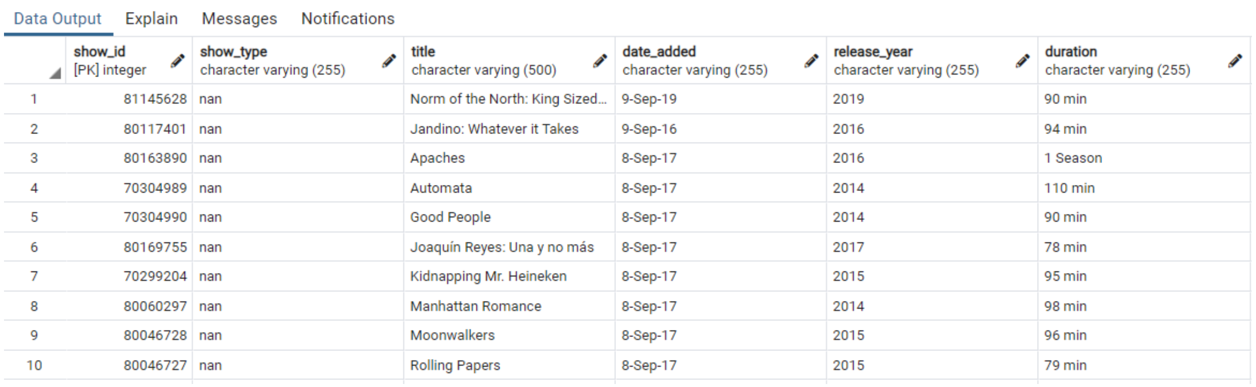


```
In [25]:  netflix_titles_Cast = pd.DataFrame ({"Titles": df_new["show_id"],
                                         "Cast": df_new["cast"]})
          netflix_titles_Cast.Cast = netflix_titles_Cast.Cast.str.split(', ')
          netflix_titles_Cast = netflix_titles_Cast.explode('Cast').reset_index(drop=True)
          netflix_titles_Cast.dropna(subset=['Cast'], inplace=True)

In [26]:  netflix_titles_Cast["Cast"] = netflix_titles_Cast["Cast"].str.replace("'","''")

In [27]:  netflix_titles_Cast.drop_duplicates (subset = ['Titles','Cast'], inplace = True)
          netflix_titles_Cast.head()

Out[27]:
```

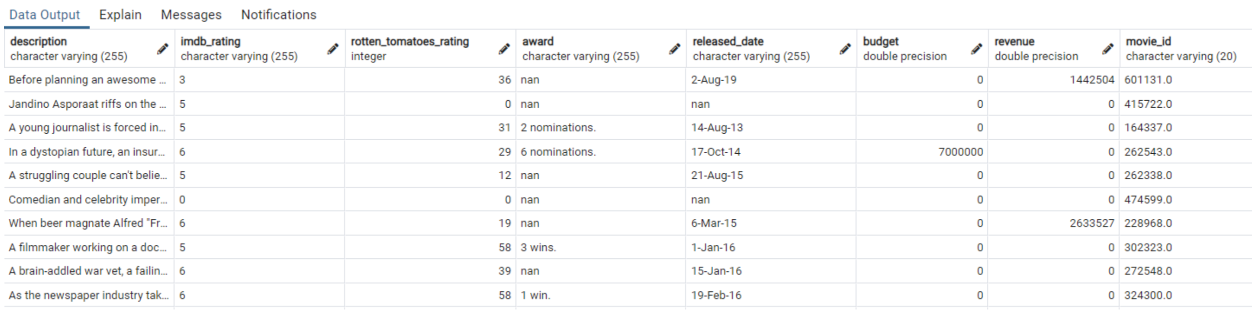
	Titles	Cast
0	81145628	Alan Marriott
1	81145628	Andrew Toth
2	81145628	Brian Dobson
3	81145628	Cole Howard
4	81145628	Jennifer Cameron

```
In [32]:  player_name_list = list(netflix_titles_Cast['Cast'])
          show_id_list = list(netflix_titles_Cast['Titles'])
          player_tile_statements = []
          for i in range(len(player_name_list)):
              statement_player_id = engine.execute(f"select player_id from players where player_name = '{player_name_list[i]}'").fetcha
              player_tile_statements.append (f"INSERT INTO player_title (player_id, show_id) values ({statement_player_id[0][0]},{show_
              print (i, " - ", player_tile_statements[i])

          24198  -  INSERT INTO player_title (player_id, show_id) values (16287,70136122)
          24199  -  INSERT INTO player_title (player_id, show_id) values (16288,70136122)
          24200  -  INSERT INTO player_title (player_id, show_id) values (4849,70136122)
          24201  -  INSERT INTO player_title (player_id, show_id) values (13881,70136122)
          24202  -  INSERT INTO player_title (player_id, show_id) values (3013,70136122)
          24203  -  INSERT INTO player_title (player_id, show_id) values (16289,70136122)
          24204  -  INSERT INTO player_title (player_id, show_id) values (13974,70136122)
          24205  -  INSERT INTO player_title (player_id, show_id) values (9072,70136122)
          24206  -  INSERT INTO player_title (player_id, show_id) values (3973,70136122)
          24207  -  INSERT INTO player_title (player_id, show_id) values (16290,80005756)
          24208  -  INSERT INTO player_title (player_id, show_id) values (16291,80005756)
          24209  -  INSERT INTO player_title (player_id, show_id) values (16292,80005756)
          24210  -  INSERT INTO player_title (player_id, show_id) values (16293,80005756)
          24211  -  INSERT INTO player_title (player_id, show_id) values (16294,80005756)
          24212  -  INSERT INTO player_title (player_id, show_id) values (2986,70153404)
          24213  -  INSERT INTO player_title (player_id, show_id) values (2906,70153404)
          24214  -  INSERT INTO player_title (player_id, show_id) values (15620,70153404)
          24215  -  INSERT INTO player_title (player_id, show_id) values (2732,70153404)
          24216  -  INSERT INTO player_title (player_id, show_id) values (16295,70153404)
          24217  -  INSERT INTO player_title (player_id, show_id) values (2272,70153404)

In [33]:  for i in range(len(player_tile_statements)):
              conn.execute(player_tile_statements[i])
```

"Title" Table:

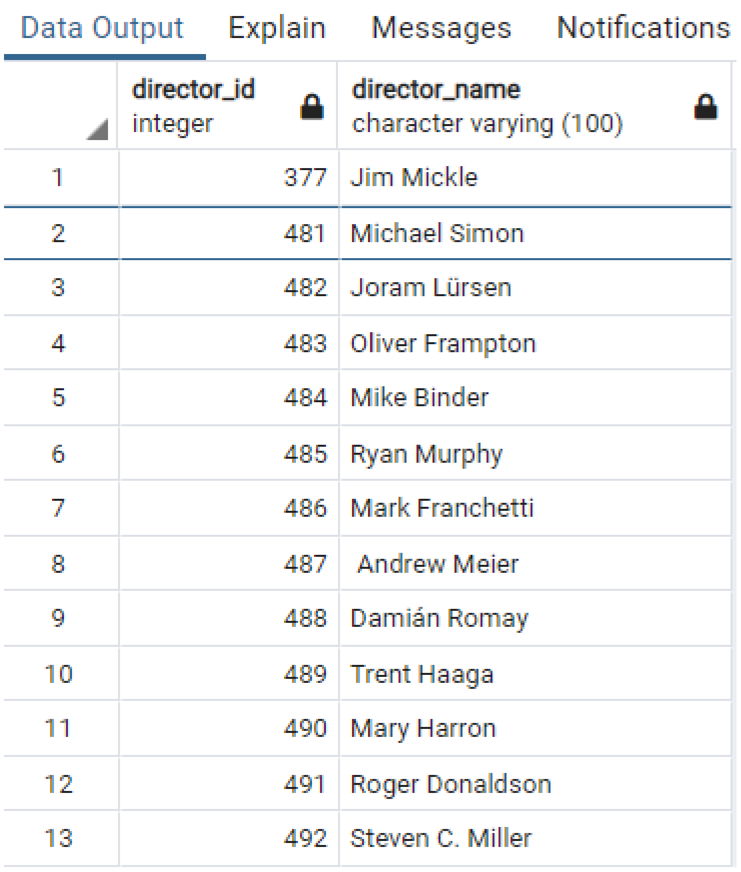


	show_id [PK] integer	show_type character varying (255)	title character varying (500)	date_added character varying (255)	release_year character varying (255)	duration character varying (255)
1	81145628	nan	Norm of the North: King Sized...	9-Sep-19	2019	90 min
2	80117401	nan	Jandino: Whatever it Takes	9-Sep-16	2016	94 min
3	80163890	nan	Apaches	8-Sep-17	2016	1 Season
4	70304989	nan	Automata	8-Sep-17	2014	110 min
5	70304990	nan	Good People	8-Sep-17	2014	90 min
6	80169755	nan	Joaquín Reyes: Una y no más	8-Sep-17	2017	78 min
7	70299204	nan	Kidnapping Mr. Heineken	8-Sep-17	2015	95 min
8	80060297	nan	Manhattan Romance	8-Sep-17	2014	98 min
9	80046728	nan	Moonwalkers	8-Sep-17	2015	96 min
10	80046727	nan	Rolling Papers	8-Sep-17	2015	79 min


Data Output | Explain | Messages | Notifications

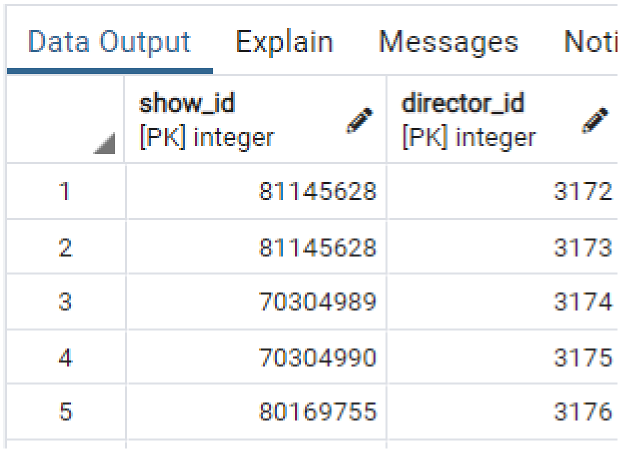
description character varying (255)	imdb_rating character varying (255)	rotten_tomatoes_rating integer	award character varying (255)	released_date character varying (255)	budget double precision	revenue double precision	movie_id character varying (20)
Before planning an awesome ...	3	36	nan	2-Aug-19	0	1442504	601131.0
Jandino Asporaat riffs on the ...	5	0	nan	nan	0	0	415722.0
A young journalist is forced in...	5	31	2 nominations.	14-Aug-13	0	0	164337.0
In a dystopian future, an insur...	6	29	6 nominations.	17-Oct-14	7000000	0	262543.0
A struggling couple can't belie...	5	12	nan	21-Aug-15	0	0	262338.0
Comedian and celebrity imper...	0	0	nan	nan	0	0	474599.0
When beer magnate Alfred "Fr...	6	19	nan	6-Mar-15	0	2633527	228968.0
A filmmaker working on a doc...	5	58	3 wins.	1-Jan-16	0	0	302323.0
A brain-addled war vet, a failin...	6	39	nan	15-Jan-16	0	0	272548.0
As the newspaper industry tak...	6	58	1 win.	19-Feb-16	0	0	324300.0

"Director" Table:


Data Output | Explain | Messages | Notifications

	director_id integer	director_name character varying (100)
1	377	Jim Mickle
2	481	Michael Simon
3	482	Joram Lürsen
4	483	Oliver Frampton
5	484	Mike Binder
6	485	Ryan Murphy
7	486	Mark Franchetti
8	487	Andrew Meier
9	488	Damián Romay
10	489	Trent Haaga
11	490	Mary Harron
12	491	Roger Donaldson
13	492	Steven C. Miller

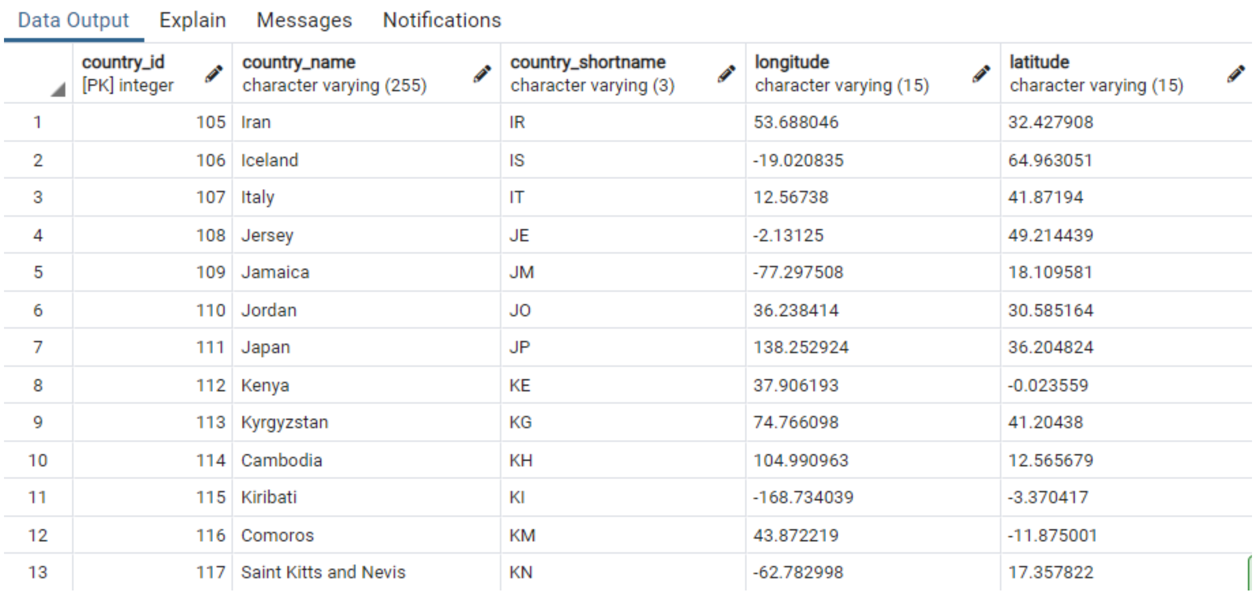
“Director_title” Table:



Data Output | Explain | Messages | Noti

	show_id [PK] integer	director_id [PK] integer
1	81145628	3172
2	81145628	3173
3	70304989	3174
4	70304990	3175
5	80169755	3176

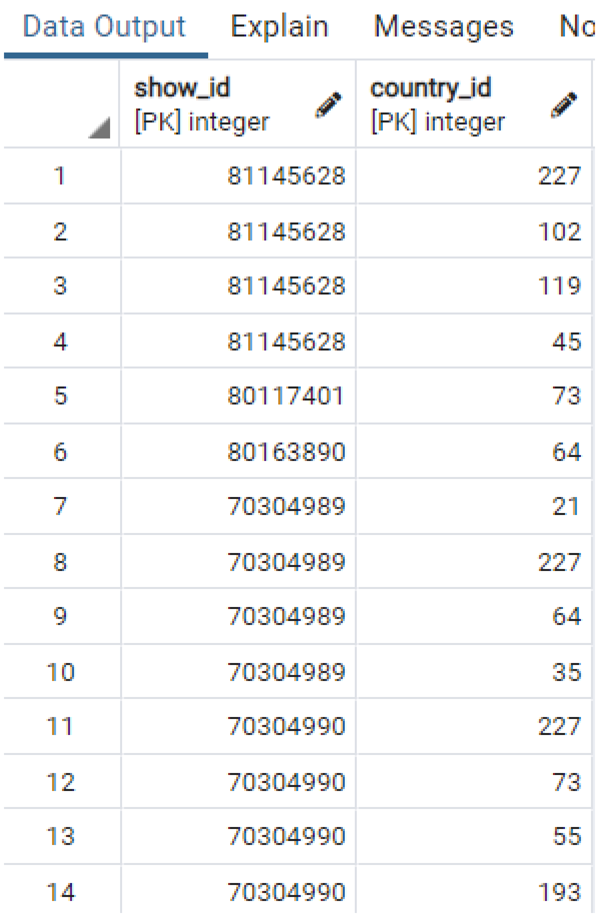
“Country” Table:



Data Output | Explain | Messages | Notifications

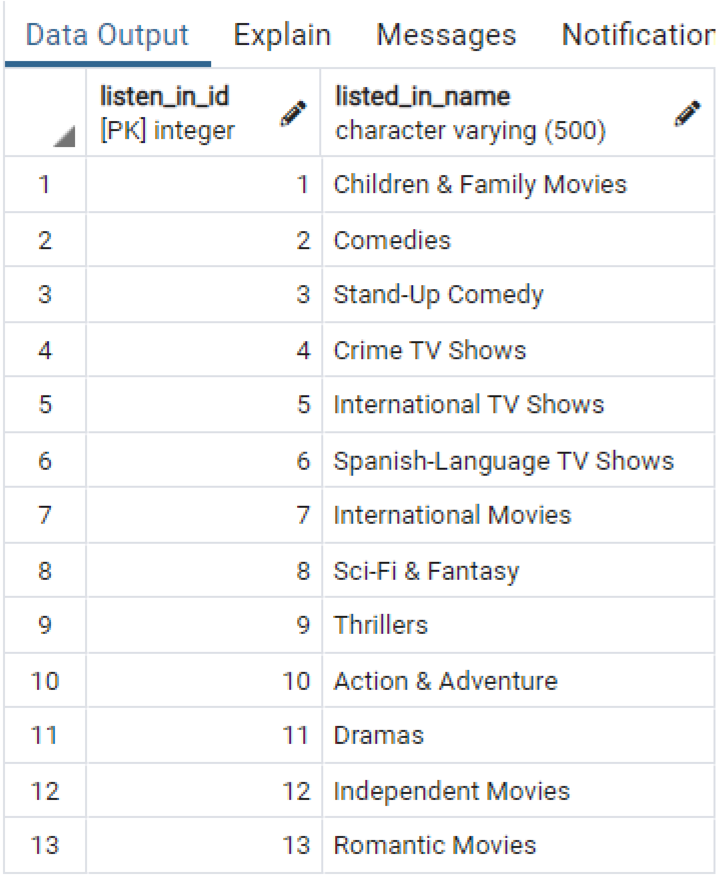
	country_id [PK] integer	country_name character varying (255)	country_shortname character varying (3)	longitude character varying (15)	latitude character varying (15)
1	105	Iran	IR	53.688046	32.427908
2	106	Iceland	IS	-19.020835	64.963051
3	107	Italy	IT	12.56738	41.87194
4	108	Jersey	JE	-2.13125	49.214439
5	109	Jamaica	JM	-77.297508	18.109581
6	110	Jordan	JO	36.238414	30.585164
7	111	Japan	JP	138.252924	36.204824
8	112	Kenya	KE	37.906193	-0.023559
9	113	Kyrgyzstan	KG	74.766098	41.20438
10	114	Cambodia	KH	104.990963	12.565679
11	115	Kiribati	KI	-168.734039	-3.370417
12	116	Comoros	KM	43.872219	-11.875001
13	117	Saint Kitts and Nevis	KN	-62.782998	17.357822

“Title_Country” Table:



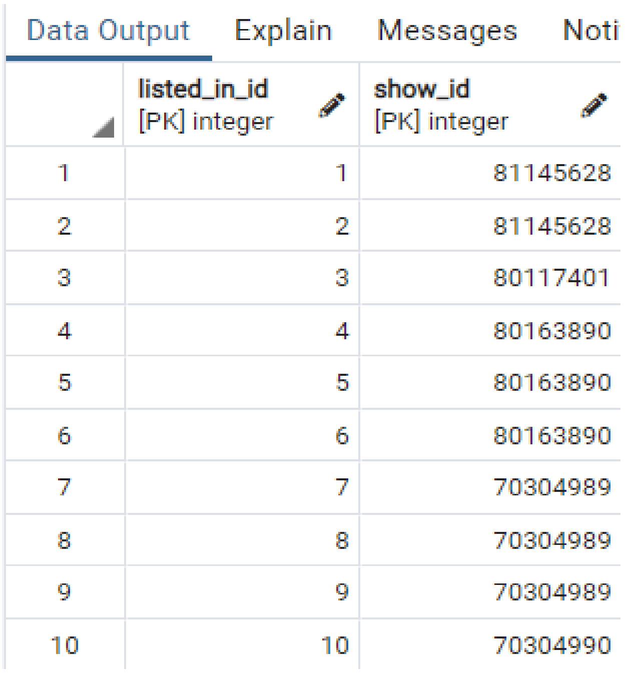
	show_id [PK] integer	country_id [PK] integer
1	81145628	227
2	81145628	102
3	81145628	119
4	81145628	45
5	80117401	73
6	80163890	64
7	70304989	21
8	70304989	227
9	70304989	64
10	70304989	35
11	70304990	227
12	70304990	73
13	70304990	55
14	70304990	193

“Listed_in” Table:



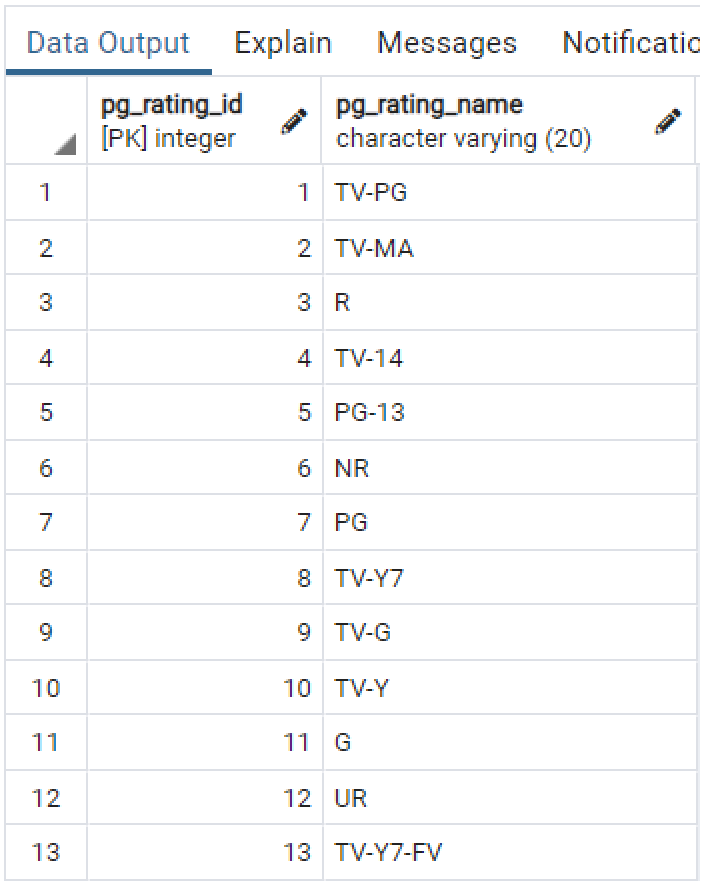
	listen_in_id [PK] integer	listed_in_name character varying (500)
1	1	Children & Family Movies
2	2	Comedies
3	3	Stand-Up Comedy
4	4	Crime TV Shows
5	5	International TV Shows
6	6	Spanish-Language TV Shows
7	7	International Movies
8	8	Sci-Fi & Fantasy
9	9	Thrillers
10	10	Action & Adventure
11	11	Dramas
12	12	Independent Movies
13	13	Romantic Movies

“Listed_in_title” Table:



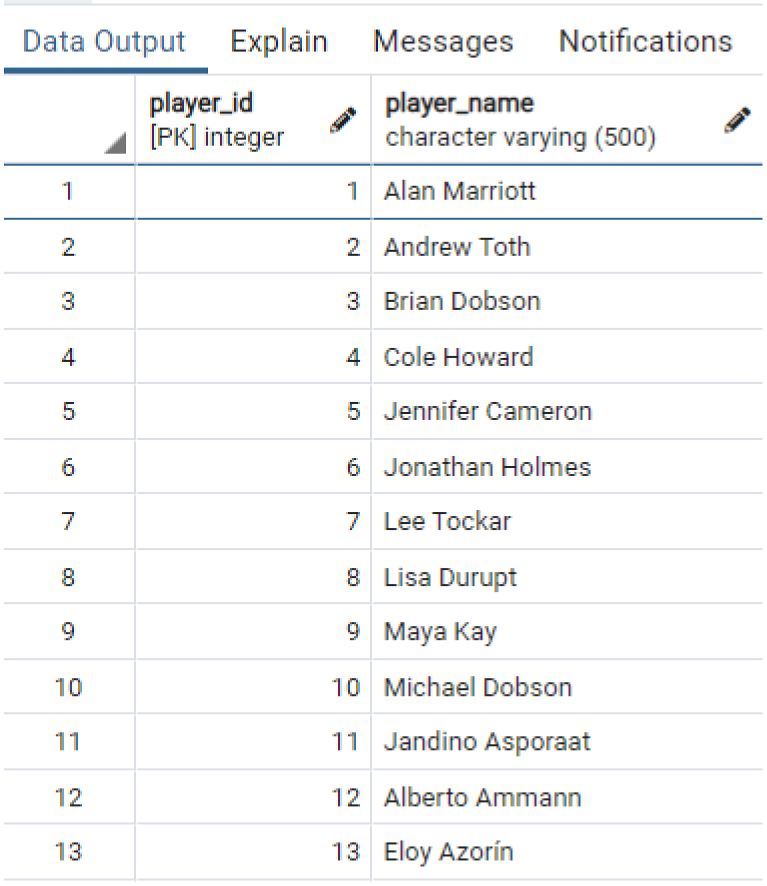
	listed_in_id [PK] integer	show_id [PK] integer
1	1	81145628
2	2	81145628
3	3	80117401
4	4	80163890
5	5	80163890
6	6	80163890
7	7	70304989
8	8	70304989
9	9	70304989
10	10	70304990

“PG_Rating” Table:



	pg_rating_id [PK] integer	pg_rating_name character varying (20)
1	1	TV-PG
2	2	TV-MA
3	3	R
4	4	TV-14
5	5	PG-13
6	6	NR
7	7	PG
8	8	TV-Y7
9	9	TV-G
10	10	TV-Y
11	11	G
12	12	UR
13	13	TV-Y7-FV

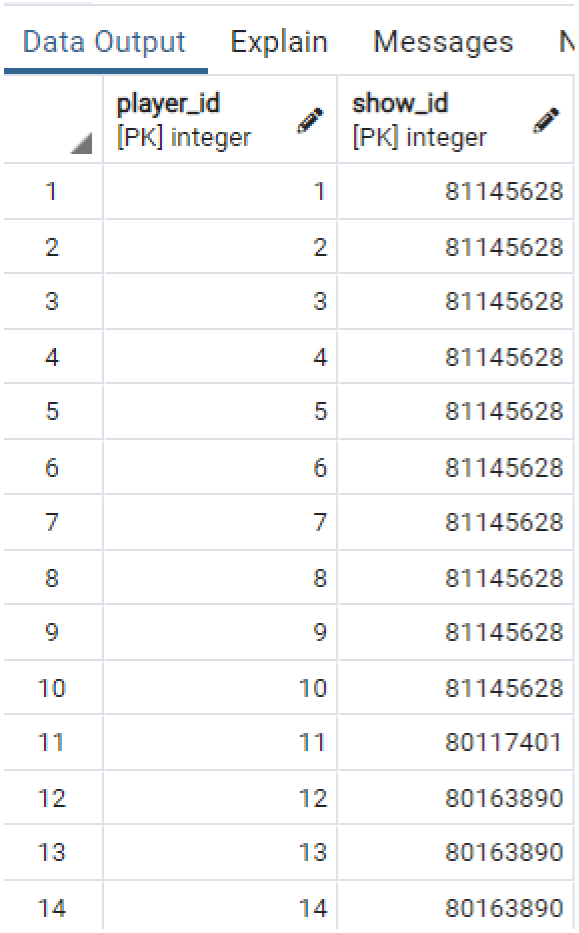
“Players” Table:



Data Output | Explain | Messages | Notifications

	player_id [PK] integer	player_name character varying (500)
1	1	Alan Marriott
2	2	Andrew Toth
3	3	Brian Dobson
4	4	Cole Howard
5	5	Jennifer Cameron
6	6	Jonathan Holmes
7	7	Lee Tockar
8	8	Lisa Durupt
9	9	Maya Kay
10	10	Michael Dobson
11	11	Jandino Asporaat
12	12	Alberto Ammann
13	13	Eloy Azorín

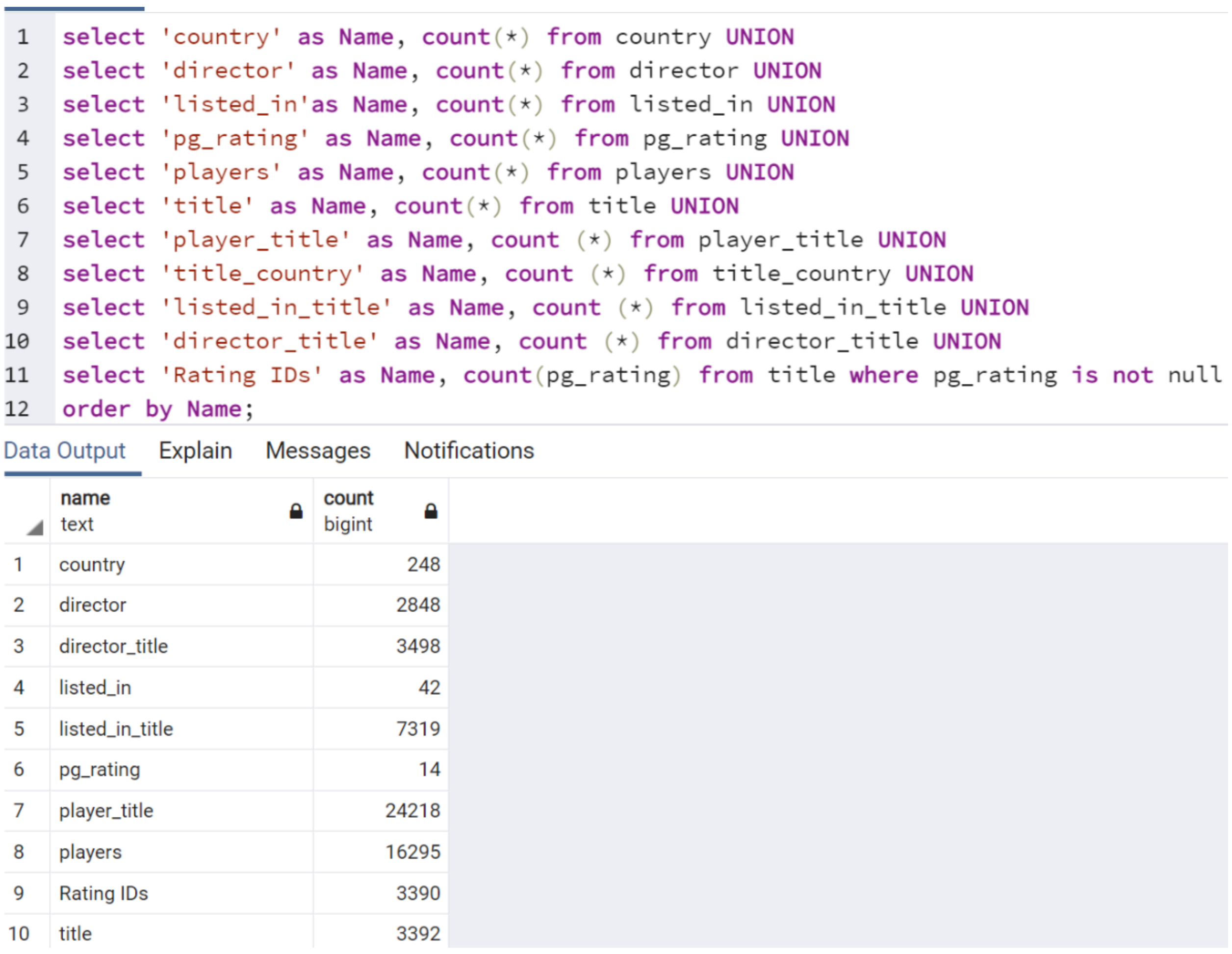
“Player_Title” Table:



	player_id [PK] integer	show_id [PK] integer
1	1	81145628
2	2	81145628
3	3	81145628
4	4	81145628
5	5	81145628
6	6	81145628
7	7	81145628
8	8	81145628
9	9	81145628
10	10	81145628
11	11	80117401
12	12	80163890
13	13	80163890
14	14	80163890

Final Database

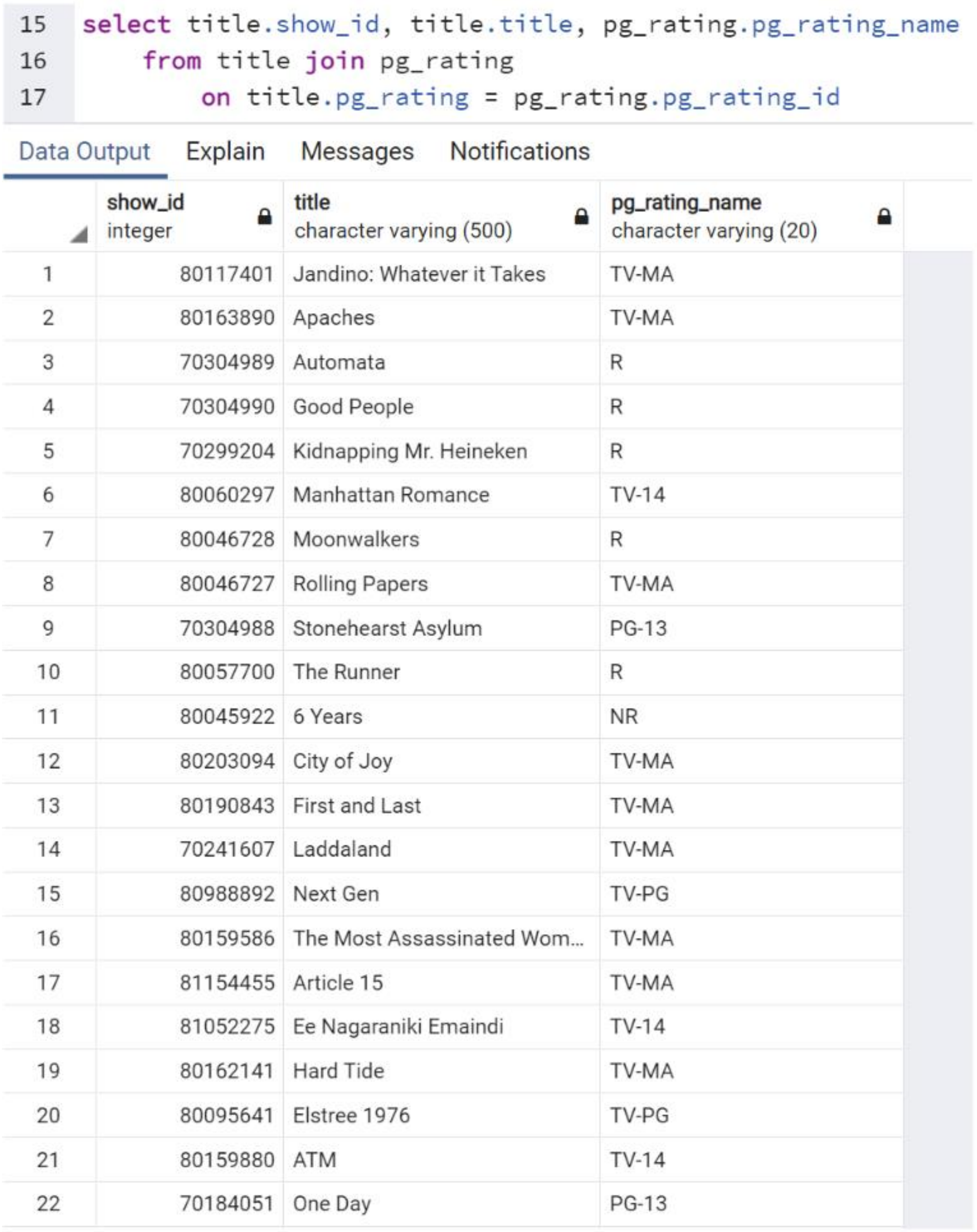
The final database contains 10 tables with the specified relations. The data base contained more than 24000 records of Netflix data in total.



```sql
select 'country' as Name, count(*) from country UNION
select 'director' as Name, count(*) from director UNION
select 'listed_in'as Name, count(*) from listed_in UNION
select 'pg_rating' as Name, count(*) from pg_rating UNION
select 'players' as Name, count(*) from players UNION
select 'title' as Name, count(*) from title UNION
select 'player_title' as Name, count (*) from player_title UNION
select 'title_country' as Name, count (*) from title_country UNION
select 'listed_in_title' as Name, count (*) from listed_in_title UNION
select 'director_title' as Name, count (*) from director_title UNION
select 'Rating IDs' as Name, count(pg_rating) from title where pg_rating is not null
order by Name;
```

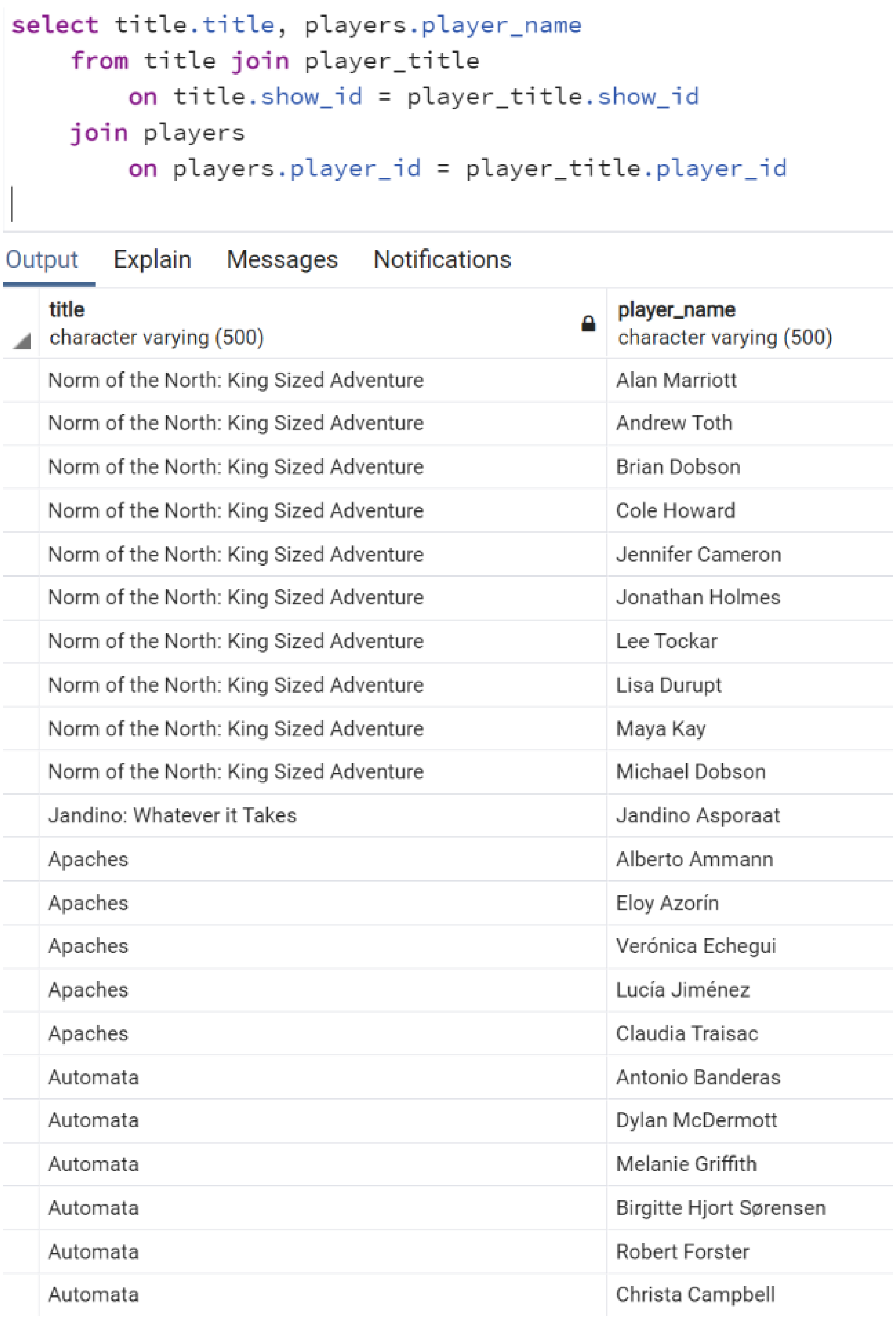
	name text	count bigint
1	country	248
2	director	2848
3	director_title	3498
4	listed_in	42
5	listed_in_title	7319
6	pg_rating	14
7	player_title	24218
8	players	16295
9	Rating IDs	3390
10	title	3392

Query Examples:



```sql
select title.show_id, title.title, pg_rating.pg_rating_name
    from title join pg_rating
        on title.pg_rating = pg_rating.pg_rating_id
```

	show_id integer	title character varying (500)	pg_rating_name character varying (20)
1	80117401	Jandino: Whatever it Takes	TV-MA
2	80163890	Apaches	TV-MA
3	70304989	Automata	R
4	70304990	Good People	R
5	70299204	Kidnapping Mr. Heineken	R
6	80060297	Manhattan Romance	TV-14
7	80046728	Moonwalkers	R
8	80046727	Rolling Papers	TV-MA
9	70304988	Stonehearst Asylum	PG-13
10	80057700	The Runner	R
11	80045922	6 Years	NR
12	80203094	City of Joy	TV-MA
13	80190843	First and Last	TV-MA
14	70241607	Laddaland	TV-MA
15	80988892	Next Gen	TV-PG
16	80159586	The Most Assassinated Wom...	TV-MA
17	81154455	Article 15	TV-MA
18	81052275	Ee Nagaraniki Emaindi	TV-14
19	80162141	Hard Tide	TV-MA
20	80095641	Elstree 1976	TV-PG
21	80159880	ATM	TV-14
22	70184051	One Day	PG-13



```
select title.title, players.player_name
    from title join player_title
        on title.show_id = player_title.show_id
    join players
        on players.player_id = player_title.player_id
```

title character varying (500)	player_name character varying (500)
Norm of the North: King Sized Adventure	Alan Marriott
Norm of the North: King Sized Adventure	Andrew Toth
Norm of the North: King Sized Adventure	Brian Dobson
Norm of the North: King Sized Adventure	Cole Howard
Norm of the North: King Sized Adventure	Jennifer Cameron
Norm of the North: King Sized Adventure	Jonathan Holmes
Norm of the North: King Sized Adventure	Lee Tockar
Norm of the North: King Sized Adventure	Lisa Durupt
Norm of the North: King Sized Adventure	Maya Kay
Norm of the North: King Sized Adventure	Michael Dobson
Jandino: Whatever it Takes	Jandino Asporaat
Apaches	Alberto Ammann
Apaches	Eloy Azorín
Apaches	Verónica Echegui
Apaches	Lucía Jiménez
Apaches	Claudia Traisac
Automata	Antonio Banderas
Automata	Dylan McDermott
Automata	Melanie Griffith
Automata	Birgitte Hjort Sørensen
Automata	Robert Forster
Automata	Christa Campbell