DMQL Project report Milestone 1

October 2022

1 Project details

Project name: Hollytics: Hollywood Analytics and Insights.
Team:

- Amardhruva Narasimha Prabhu (amardhru)
- Himani Madan (himanima)
- Sarthak Jain (sjain34)

2 Problem Statement

The Hollywood data is monopolised by large corporations like Amazon. If someone wants quick insights on the rising trends in Hollywood, it is not possible. Even though sites like IMDB provide their database dumps, there aren't viable solutions to analyze the Hollywood data to get reliable insights. For example, getting the list of best directors, popular genre, movie's ratings and writers information. Hence, we selected this database to resolve such problems.

2.1 Approach to solve the problem

We used PostgreSQL to load IMDB dataset into database and perform analysis over the dataset to get insights. We made suitable modifications in the TSV format of the dataset and decomposed it into Boyce - Codd Normal Form(BCNF) form. We constructed the SQL queries which can be used to generate the analytics over the dataset. We created a Python script to load the 3GB data from original TSV format to PostgreSQL table format. We aim to create a web based tool which parses the IMDB dumps and produces valuable insights which may help an new actors and producers. We generate insights like the trending genres, popular directors and writers for the titles by using SQL queries. We put this on a Web based front-end where everyone can access the analysis and can gather valuable insights without knowing any Technical knowledge.

2.2 Deliverables

- A script to load the imdb dump into a database.
- A set of queries to gather valuable insights from the database.
- Web interface which automatically runs the queries and generates insights.

2.3 Why Database?

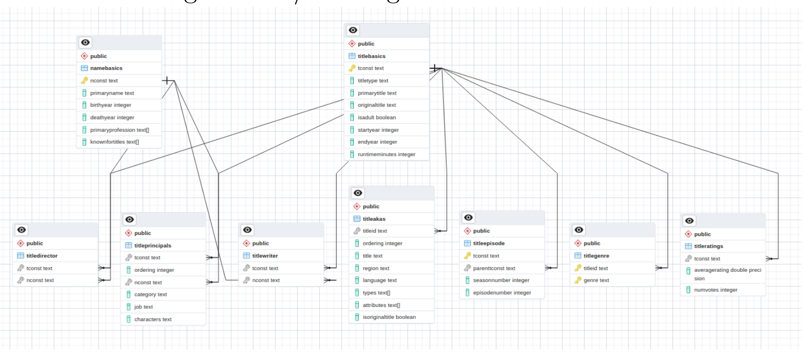
We could store the Hollywood data on a spreadsheet program. But the links between the relations are exceedingly complex and we must use a database to reliably store the links between the tables and run complex queries on them.

3 Target user

The target audience are up and coming actors, directors, producers and business executives who are interested in fresh talent. They need quick and valuable analysis into the trends which shift constantly in the entertainment industry. By having suitable insights the customer can make better decisions than the competition and generate revenue.

4 E-R Diagram

Figure 1: E/R Diragram for the database



5 Tasks

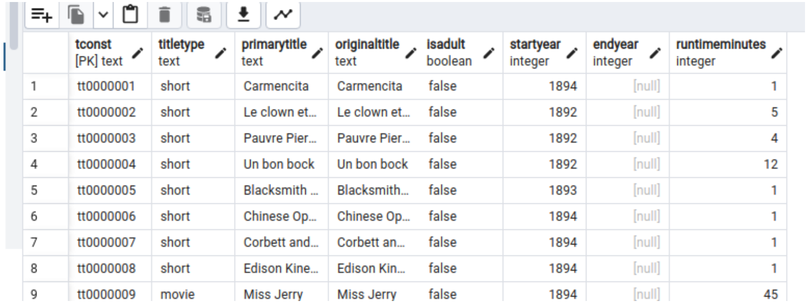
5.1 Selecting usecase domain

Our project can be utilized in entertainment industry specifically hollywood. We selected this domain because there are enormous number of movies, TV series etc. and very few resources which can provide proper analysis of this data. Since the data is huge, the insights and analytics would become more interesting on this. Actors, directors, audience, writers etc can use our project to get their desired information.

5.2 Creating Sample database by hand

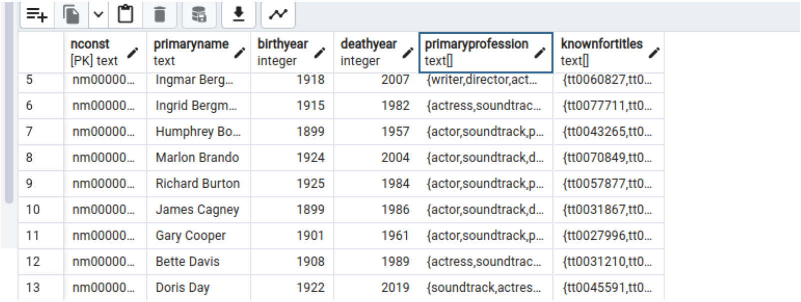
We validate the database schema by creating a sample database by hand. We manually pick a few rows in the IMDB dumps as sample data and we load it into the database.

Figure 2: Sample data 1



	tconst [PK] text	titletype text	primarytitle text	originaltitle text	isadult boolean	startyear integer	endyear integer	runtimeminutes integer
1	tt0000001	short	Carmencita	Carmencita	false	1894	[null]	1
2	tt0000002	short	Le clown et...	Le clown et...	false	1892	[null]	5
3	tt0000003	short	Pauvre Pier...	Pauvre Pier...	false	1892	[null]	4
4	tt0000004	short	Un bon bock	Un bon bock	false	1892	[null]	12
5	tt0000005	short	Blacksmith ...	Blacksmith...	false	1893	[null]	1
6	tt0000006	short	Chinese Op...	Chinese Op...	false	1894	[null]	1
7	tt0000007	short	Corbett and...	Corbett an...	false	1894	[null]	1
8	tt0000008	short	Edison Kine...	Edison Kin...	false	1894	[null]	1
9	tt0000009	movie	Miss Jerry	Miss Jerry	false	1894	[null]	45

Figure 3: Sample data 2



	nconst [PK] text	primaryname text	birthyear integer	deathyear integer	primaryprofession text[]	knownfortitles text[]
5	nm00000...	Ingmar Berg...	1918	2007	{writer,director,act...	{tt0060827,tt0...
6	nm00000...	Ingrid Bergm...	1915	1982	{actress,soundtrac...	{tt0077711,tt0...
7	nm00000...	Humphrey Bo...	1899	1957	{actor,soundtrack,p...	{tt0043265,tt0...
8	nm00000...	Marlon Brando	1924	2004	{actor,soundtrack,d...	{tt0070849,tt0...
9	nm00000...	Richard Burton	1925	1984	{actor,soundtrack,p...	{tt0057877,tt0...
10	nm00000...	James Cagney	1899	1986	{actor,soundtrack,d...	{tt0031867,tt0...
11	nm00000...	Gary Cooper	1901	1961	{actor,soundtrack,p...	{tt0027996,tt0...
12	nm00000...	Bette Davis	1908	1989	{actress,soundtrac...	{tt0031210,tt0...
13	nm00000...	Doris Day	1922	2019	{soundtrack,actres...	{tt0045591,tt0...

5.3 Design database schema

We have 9 relations in the database listed below with their details:

- titlebasics: This relation has the information for 8 attributes related to title. Primary key in this relation is tconst. Attributes are listed below:

    1. tconst (string): title is provided with an alphanumeric unique identifier tconst.
    2. titleType (string): the type of the title (e.g. movie, tvseries, tvepisode)

3. primaryTitle (string): the title used by the filmmakers on promotional materials at the point of release. This can be considered as the more popular title.
4. originalTitle (string): original title, in the original language.
5. isAdult (boolean): value 1 for adult title and 0 for non-adult title.
6. startYear (YYYY): represents the release year of a title. In the case of TV Series, it is the series start year.
7. endYear (YYYY): TV Series end year. Null for all other title types.
8. runtimeMinutes(int): primary runtime of the title, in minutes.

- titleakas: This relation contains 8 attributes which collectively define all the information for title. Primary key of this relation is the combination of titleid and ordering and foreign key is titleid. Attributes in this table are:

  1. titleid(text): title is provided with an alphanumeric unique identifier titleid.
  2. ordering(int): titleid can be repeated in this relation therefore, to identify every tuple uniquely, there is ordering number for every same titleid.
  3. title(text): It represents the localized title of the movie.
  4. region(text): It has the value of the region of this version of the title.
  5. language(text): This attribute represents the language of the title.
  6. types(text array): Enumerated set of attributes for this alternative title. One or more of the following: ”alternative”, ”dvd”, ”festival”, ”tv”, ”video”, ”working”, ”original”, ”imdbDisplay”.
  7. attributes(text array): Additional terms to describe this alternative title, not enumerated.
  8. isOriginalTitle(boolean) : value for this attribute is 1 for original title and 0 for not original title.

- titleepisode: It contains the information for the Tv episodes and has 4 attributes with tconst as primary key and parentTconst as foreign key and the referenced key is tconst of titlebasic relation. Attributes are:

  1. tconst (text): episode is provided with an alphanumeric unique identifier tconst.
  2. parentTconst(text): alphanumeric id of the TV episode
  3. seasonNumber(int): season number of the TV series
  4. episodeNumber(int): episode number of the tconst in a season

- namebasics: This relation contains the details about names of the people involved in hollywood in the following attributes:

1. nconst (string): title is provided with an alphanumeric unique identifier name/person.
2. primaryName (string): most often credited name of the person.
3. birthYear(integer): year of birth in 'YYYY' format.
4. deathYear(integer): year of death of the person in 'YYYY' format if applicable, else null.
5. primaryProfession (array of text): the top-3 professions of the person because a person can be involved in many professions.
6. knownForTitles (array of text): titles of the movies/series for which the person is well-known.

- titleprincipals: This relation represents the principal cast/crew for titles and has two foreign keys, tconst referenced from table tbasic and nconst referenced from table namebasics. Following are the 6 attributes of the table.

  1. tconst (string): title is provided with an alphanumeric unique identifier tconst.
  2. ordering (integer): a number to uniquely identify rows for a given titleId.
  3. nconst (string): person is provided with an alphanumeric unique identifier nconst
  4. category (string): the category of job that person was in
  5. job (string): the specific job title if applicable, else null
  6. characters (string): the name of the character played if applicable, else null

- titlewriter: represents the information of writers for every title.Primary key for the relation is the combination of nconst and tconst. tconst is the foreign key referenced from table titlebasics and nconst is the foreign key referenced from table namebasics. Cascade deletion is turned on the both foreign keys. Following are the attributes:

  1. nconst(text): contains the alphanumeric identifier for writer name
  2. tconst(text): alphanumeric identifier for title.

- titledirector: contains the details of director for every title.Primary key for the relation is the combination of nconst and tconst. tconst is the foreign key referenced from table titlebasics and nconst is the foreign key referenced from table namebasics. Cascade deletion is turned on the both foreign keys. Following are the attributes:

  1. nconst(text): director is provided with an alphanumeric unique identifier nconst.

2. tconst(text): title is provided with an alphanumeric unique identifier tconst.

- titlegenre: contains the genre for the titles. One titleid can have multiple genre. Primary key in table is the combination of titleid and genre. TitleId is the foreign key referenced from relation titleakas. Following are the attributes of titlegenre:

  1. titleid(text): contains the id for the title.
  2. genre(text): represents the genre of the title.

- titleratings: contains the ratings for the titles. tconst is the foreign key referenced from table titlebasics. Following are the attributes:

  1. tconst (text): title is provided with an alphanumeric unique identifier tconst.
  2. averageRating(int): weighted average of all the individual user ratings.
  3. numVotes(int): number of votes the title has received.

5.4 Acquire large production dataset

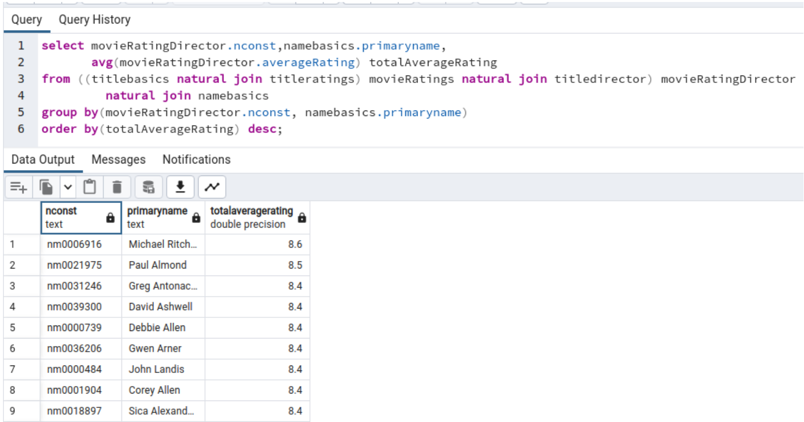
Actual production data for IMDB dataset can be found on the link `datasets.imdbws.com`. Dataset is divided into multiple TSV format files contained in compressed folders and is of around 3GB. This 3GB of data provides us more than 10,000 tuples(even more) easily. For loading of the dataset into our relation schema, we used a Python script. We also performed the cleaning of dataset e.g we replaced the `\N` in data values by null as it was placed for empty values. We used psycopg2 in python to connect to postgreSQL database and used its object further to access rows from TSV file. Finally, we dump the data into the postgreSQL by committing the pyscopg2 cursor object.

5.5 Run SELECT queries

We executed 4 different queries on our database.

- We displayed the results for highest rated directors to the lowest rated directors. This result can be utilized by the writers, actors or other cast crew staff to choose whether they want to work with a specific director or not based on director's average rating.

Figure 4: Highest rated Directors

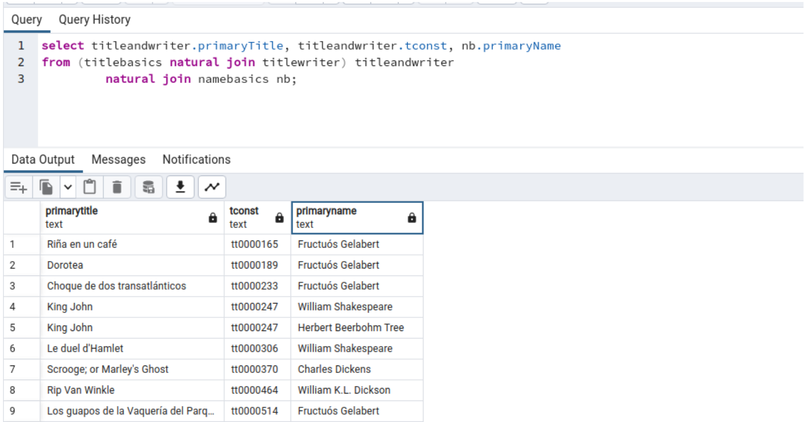


```
select movieRatingDirector.nconst,namebasics.primaryname,
       avg(movieRatingDirector.averageRating) totalAverageRating
from ((titlebasics natural join titleratings) movieRatings natural join titledirector) movieRatingDirector
         natural join namebasics
group by(movieRatingDirector.nconst, namebasics.primaryname)
order by(totalAverageRating) desc;
```

	nconst text	primaryname text	totalaveragerating double precision
1	nm0006916	Michael Ritch...	8.6
2	nm0021975	Paul Almond	8.5
3	nm0031246	Greg Antonac...	8.4
4	nm0039300	David Ashwell	8.4
5	nm0000739	Debbie Allen	8.4
6	nm0036206	Gwen Arner	8.4
7	nm0000484	John Landis	8.4
8	nm0001904	Corey Allen	8.4
9	nm0018897	Sica Alexand...	8.4

- Second query, represents the list of titles and title's writer. This can be utilize to view who was the writer of a specific title. Attributes that we have displayed in this query are primaryName of the writer by which he is well known, name of the title and unique identifier of the title.

Figure 5: Information about writers

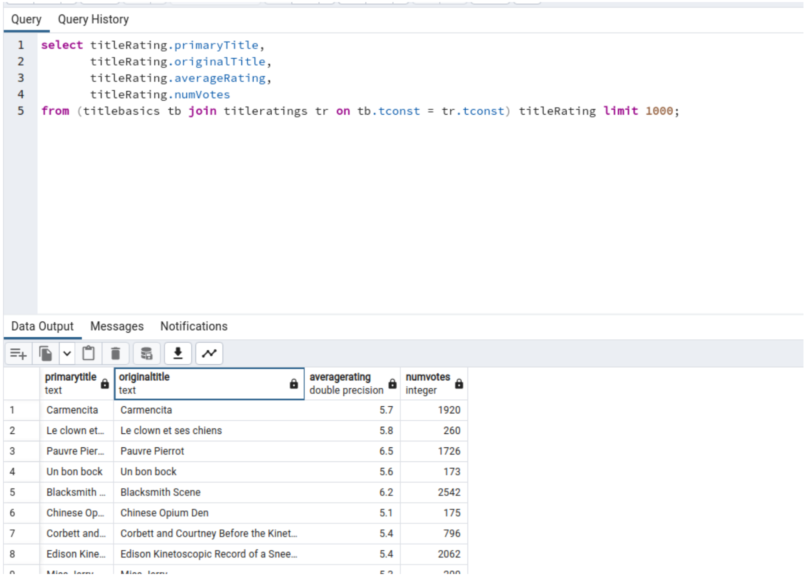


```
select titleandwriter.primaryTitle, titleandwriter.tconst, nb.primaryName
from (titlebasics natural join titlewriter) titleandwriter
         natural join namebasics nb;
```

	primarytitle text	tconst text	primaryname text
1	Riña en un café	tt0000165	Fructuós Gelabert
2	Dorotea	tt0000189	Fructuós Gelabert
3	Choque de dos transatlánticos	tt0000233	Fructuós Gelabert
4	King John	tt0000247	William Shakespeare
5	King John	tt0000247	Herbert Beerbohm Tree
6	Le duel d'Hamlet	tt0000306	William Shakespeare
7	Scrooge; or Marley's Ghost	tt0000370	Charles Dickens
8	Rip Van Winkle	tt0000464	William K.L. Dickson
9	Los guapos de la Vaquería del Parq...	tt0000514	Fructuós Gelabert

- This query represents the average rating for the titles of movies/TV series with the number of votes the title has received as numVotes. AverageRating is the weighted average of the rating received from individuals.

Figure 6: Average movie rating

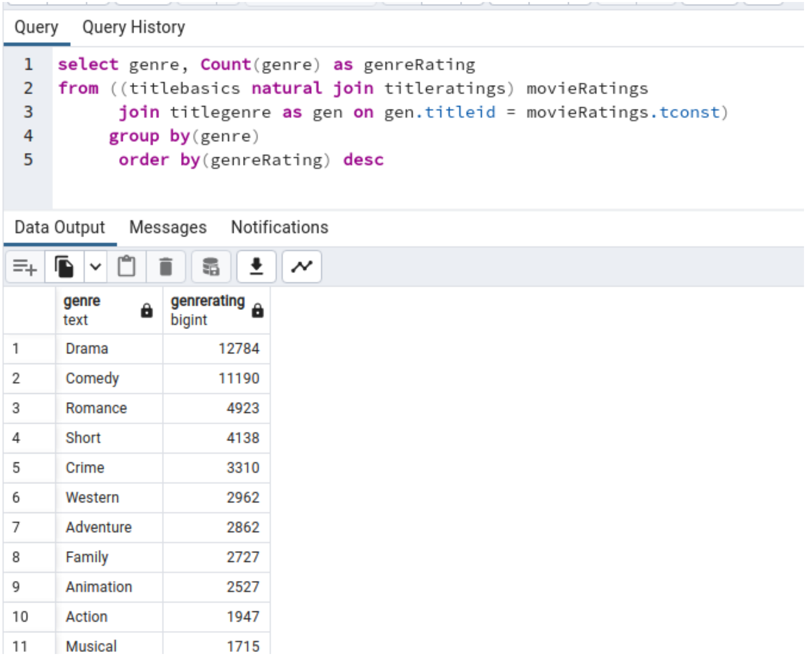


```sql
select titleRating.primaryTitle,
       titleRating.originalTitle,
       titleRating.averageRating,
       titleRating.numVotes
from (titlebasics tb join titleratings tr on tb.tconst = tr.tconst) titleRating limit 1000;
```

	primarytitle text	originaltitle text	averagerating double precision	numvotes integer
1	Carmencita	Carmencita	5.7	1920
2	Le clown et...	Le clown et ses chiens	5.8	260
3	Pauvre Pier...	Pauvre Pierrot	6.5	1726
4	Un bon bock	Un bon bock	5.6	173
5	Blacksmith ...	Blacksmith Scene	6.2	2542
6	Chinese Op...	Chinese Opium Den	5.1	175
7	Corbett and...	Corbett and Courtney Before the Kinet...	5.4	796
8	Edison Kine...	Edison Kinetoscopic Record of a Snee...	5.4	2062
9	Miss Jerry	Miss Jerry	[illegible]	[illegible]

- Last query shows the results of genre with their likeness. Here, genre and genreRating means the number of times this genre's title was released. We are considering the popularity by the releasing times of a genre. Based on this, we can conclude which genre is more popular and we have listed them in a decreasing order.

Figure 7: Most liked genre



```sql
select genre, Count(genre) as genreRating
from ((titlebasics natural join titleratings) movieRatings
       join titlegenre as gen on gen.titleid = movieRatings.tconst)
      group by(genre)
       order by(genreRating) desc
```

	genre text	genrerating bigint
1	Drama	12784
2	Comedy	11190
3	Romance	4923
4	Short	4138
5	Crime	3310
6	Western	2962
7	Adventure	2862
8	Family	2727
9	Animation	2527
10	Action	1947
11	Musical	1715

6 Scraping script

We used the below python script to load the data from TSV to postgresSQL database. Psycopg2 was used to connect to the database and a cursor was created from the connection to iterate over TSV data and load to postgresSQL. PostgresSQL already had the relations created in the database by the SQL script. Python script is represented below.

```python
import json

import psycopg2
import gzip
from tqdm import tqdm

limit = 50000

with open('secret.json') as fp:
    settings = json.load(fp)

conn = psycopg2.connect(dbname="imdb", user=settings['username'], password=settings['password'], host = settings['host'], port = settings['port'])
```

```
cur = conn.cursor()

cur.execute('DROP SCHEMA IF EXISTS public CASCADE')
cur.execute('CREATE SCHEMA public')

with open('ImdbCreateSchema.sql') as fp:
    queries = fp.read()
cur.execute(queries)

def splitsanitize(val: str):
    return val.split(',') if val != '\\N' else []

with gzip.open('dataset/title.basics.tsv.gz', mode="rt") as fp:
    print(next(fp))
    for i,row in enumerate(tqdm(fp)):
        if i == limit:
            break
        row = row.strip().split('\t')
        try:

            genres = [ (row[0],i) for i in splitsanitize(row[8])]
            for key,col in enumerate(row):
                if col =='\\N':
                    row[key]=None
            del row[8]
            cur.execute('INSERT INTO titlebasics(tconst,
            ↪   titleType, primaryTitle, originalTitle,
            ↪   isAdult,startYear, endYear, runtimeMinutes)
            ↪   VALUES (%s , %s, %s, %s, %s, %s, %s, %s)',
             row)
            cur.executemany('INSERT INTO titlegenre VALUES(%s,
            ↪   %s)', genres)
        except Exception as ex:
            print(row)
            print(ex)
            raise ex

with gzip.open('dataset/title.akas.tsv.gz', mode="rt") as fp:
    print(next(fp))
    cur.execute('select max(tconst) from titlebasics')
    maxtitle = cur.fetchone()[0]
    for i,row in enumerate(tqdm(fp)):
        if i == limit:
            break
        row = row.strip().split('\t')
```

```python
        try:
            for key,col in enumerate(row):
                if col =='\\N':
                    row[key]=None
            # if row[0]>=maxtitle:
            #     continue
            cur.execute('select count(tconst) from titlebasics where tconst=%s', (row[0],))
            if cur.fetchone()[0]==0:
                continue
            cur.execute('INSERT INTO titleakas VALUES (%s , %s, %s, %s, %s, %s, %s, %s)',
             row)
        except Exception as ex:
            print(row, maxtitle)
            print(ex)
            raise ex

with gzip.open('dataset/name.basics.tsv.gz', mode="rt") as fp:
    print(next(fp))
    for i,row in enumerate(tqdm(fp)):
        if i == limit:
            break
        row = row.strip().split('\t')
        row[4] = row[4].split(',')
        row[5] = row[5].split(',')
        try:
            for key,col in enumerate(row):
                if col =='\\N':
                    row[key]=None
            cur.execute('INSERT INTO namebasics VALUES (%s , %s, %s, %s, %s, %s)',
             row)
        except Exception as ex:
            print(row)
            print(ex)
            raise ex

with gzip.open('dataset/title.crew.tsv.gz', mode="rt") as fp:
    print(next(fp))
    cur.execute('select max(tconst) from titlebasics')
    maxtitle = cur.fetchone()[0]
    cur.execute('select max(nconst) from namebasics')
    maxname = cur.fetchone()[0]
    for i,row in enumerate(tqdm(fp)):
        if i == limit:
            break
        row = row.strip().split('\t')
        if row[0]>=maxtitle:
            continue
        row[1] = row[1].split(',') if row[1] != '\\N' else []
        row[2] = row[2].split(',') if row[2] != '\\N' else []
        row[1] = list(filter(lambda x: x<maxname, row[1]))
        row[2] = list(filter(lambda x: x<maxname, row[2]))
        try:
            for key,col in enumerate(row):
                if col =='\\N':
                    row[key]=None
            cur.executemany('INSERT INTO titledirector VALUES (%s
            ↪  , %s)',
             ((row[0], i) for i in row[1]))
            cur.executemany('INSERT INTO titlewriter VALUES (%s ,
            ↪  %s)',
             ((row[0], i) for i in row[2]))
        except Exception as ex:
            print(row, maxtitle, maxname)
            print(ex)
            raise ex

with gzip.open('dataset/title.episode.tsv.gz', mode="rt") as fp:
    print(next(fp))
    failcount = 0
    for i,row in enumerate(tqdm(fp)):
        if i == limit:
            break
        row = row.strip().split('\t')
        try:
            for key,col in enumerate(row):
                if col =='\\N':
                    row[key]=None
            if row[1]>=f'tt{limit:07}':
                continue
            cur.execute('INSERT INTO titleepisode VALUES (%s ,
            ↪  %s, %s, %s)',
             row)
        except Exception as ex:
            print(row)
            print(ex)
            raise ex

with gzip.open('dataset/title.principals.tsv.gz', mode="rt") as
↪  fp:
    print(next(fp))
    failcount = 0
    cur.execute('select max(tconst) from titlebasics')
    maxtitle = cur.fetchone()[0]
    cur.execute('select max(nconst) from namebasics')
    maxname = cur.fetchone()[0]
    for i,row in enumerate(tqdm(fp)):
        if i == limit:
            break
        row = row.strip().split('\t')
        row[4] = splitsanitize(row[4])
        row[5] = splitsanitize(row[5])
        try:
            for key,col in enumerate(row):
                if col =='\\N':
                    row[key]=None
            if row[0]>=maxtitle:
                continue
            if row[2]>=maxname:
                continue
            cur.execute('INSERT INTO titleprincipals VALUES (%s ,
            ↪  %s, %s, %s, %s, %s)',
             row)
        except Exception as ex:
            print("ROW",row, maxtitle, maxname)
            print(ex)
            raise ex

with gzip.open('dataset/title.ratings.tsv.gz', mode="rt") as fp:
    print(next(fp))
    failcount = 0
    cur.execute('select max(tconst) from titlebasics')
    maxtitle = cur.fetchone()[0]
    for i,row in enumerate(tqdm(fp)):
        if i == limit:
            break
        row = row.strip().split('\t')
        try:
            for key,col in enumerate(row):
                if col =='\\N':
                    row[key]=None
            if row[0]>=maxtitle:
                continue
            cur.execute('INSERT INTO titleratings VALUES (%s ,
            ↪  %s, %s)',
             row)
        except Exception as ex:
            print("ROW",row, maxtitle)
            print(ex)
            raise ex

conn.commit()
```

7 References

https://www.postgresql.org/docs/ https://www.imdb.com/interfaces/ https://datasets.imdbws.com/
[1] Python Software Foundation. Python Language Reference, version 2.7.
Available at http://www.python.org