Università di Pisa

Large-Scale adn Multi-structured Databases - Project: Rotten Movies

Fabio Piras, Giacomo Volpi, Guillaume Quint

Contents

Chapter 1

Introduction

Rotten Movies is a web service where user can keep track of the general liking for movies, they can also share their thoughts with the world after signing up. In addition user can follow renowned top critic to be constantly updated with their latest review.

Guest user, as well as all other, can browse or search movies based on filters like: title, year of release and actor who worked in it. They can also view a Hall of Fame for the most positive review genres, production houses and years in which the best movie were released.

For this project, the application has been developed in Java with the Spring framework and as Thymeleaf as engine to implement a web GUI. The application use a document database to store the main information about the movies, user, review and actor; a graph database is instead used to keep track of the relationship between normal user and top critic in terms of who follows who and between the movie and the user who reviewed it.

Chapter 2

Feasibility Study

The very first step was to perform is to look up for what type of data we need to create the application and how to handle it by performing a feasibility study.

1 Dataset analysis and creation

Initially we searched online for available dataset, we ended up with five different files coming from Kaggle and imdb with a combined storage of 4.18 Gb:

- title_principal.tsv
- name_basic.tsv
- title_basic.tsv
- title_crew.tsv
- rotten_movies.csv
- rotten_reviews.csv

The first three were found on the imdb site containing general data about the title of the entries, type (not all were movies), cast, crew and other useful information. The last two came from Kaggle and they contain data scraped from the rotten tomatoes site regarding the movies rating and their reviews. All of these dataset were organized in a relational manner.

The next step was to delete all useless information and perform a join operation to generate one single final dataset for the document database. These operations were performed through a python script with the help of the pandas library and Google Colab, the end result was a single json file of 270 Mb. For this steps see: Appendix **??**, Appendix **??**, Appendix **??**, Appendix **??**, Appendix **??**, Appendix **??**.

The structure of the movie document is the following:

```
{
    "_id" : ObjectId(<<id_field>>),
    "primaryTitle": "The first ever movie",
    "year": 1989,
    "runtimeMinutes": 70,
    "genres": ["Crime", "Drama", "Romantic"],
    "productionCompany": "Dingo Picture Production" ,
```

```
    "personnel": [
        {
            "name": "John Doe",
            "category": "producer",
            "job": "writer"
        },
        {
            "name": "Christopher Lee",
            "category": "actor",
            "character": ["The one"]
        },
        ...
    ],
    "top_critic_fresh_count": "4",
    "top_critic_rotten_count": 0,
    "user_fresh_count": 14,
    "user_rotten_count": 1,
    "top_critic_rating": 100,
    "top_critic_status": "Fresh",
    "user_rating": 93,
    "user_status": "Fresh",
    "review":
    [
        {
            "critic_name":  "AntonyE",
            "review_date":  "2018-12-07",
            "review_type":  "Rotten",
            "top_critic": false,
            "review_content":"I really didn't liked it!",
            "review_score": "1/10"
        },
        {
            "critic_name":  "AntonyE",
            "review_date":  "2015-12-07",
            "review_type":  "Fresh",
            "top_critic": true,
            "review_content":"I really liked it!",
            "review_score": "A-"
        },
        ...
    ]
}
```

1 .1 Creation of the user collection

The following step was to generate a collection for the user, this was achieved by starting from the movie collection and for each different critic_name generating a document containing all the information necessary like: username, password, registrationDate and birthdayDate in case the user wasn't a top critic. For this steps see: Appendix ??, Appendix ??, Appendix ??.

The structure of the user document is the following:

```
{
    "_id"                : ObjectId(<<id_field>>),
```

```
    "username"          :"AntonyE",
    "password"          :"hashed_password",
    "firstName"         :"Anton",
    "lastName  "        :"Ego",
    "registrationreview_date"  :"2019-06-29",
    "review_date of birth"     :"2002-07-16",
    "last3Reviews":
        [
            {
                "_id"               : ObjectId(<<id_field>>),
                "primaryTitle"    : "Star Wars: A new Hope",
                "review_date"           : "2018-12-07",
                "review_type"     : "fresh",
                "top_critic"   : false,
                "review_content"        : "I really liked it!",
                "vote"          : "8/10"
            },
            {
                "_id"               : ObjectId(<<id_field>>),
                "primaryTitle"    : "Ratatouille",
                "review_date"           : "2019-02-17",
                "review_type"     : "rotten",
                "top_critic"   : false,
                "review_content"        : "The director was also
    ↪ controlled by a rat",
                "vote"          : "D+"
            },
            {
                "_id"               : ObjectId(<<id_field>>),
                "primaryTitle"    : "300",
                "review_date"           : "2020-02-15",
                "review_type"     : "fresh",
                "top_critic"   : false,
                "review_content"        : "Too much slow motion",
                "vote"          : "3.5/5"
            }
        ],
    "otherReview":[
        {
            "movie_id" : ObjectId(<<id_field>>),
            "primaryTitle" : "Evidence",
            "review_index": 11
        },
        ...
    ]

}
```

A Python Code

A .1 Creation of one single dataset from the tsv imdb file

```
1
2  import pandas as pd
3  from google.colab import drive
4  drive.mount('/content/drive')
5  numberofrows=None #100000
6  title_basics = pd.read_csv("/content/drive/MyDrive/Dataset/Original/
       title_basics.tsv",sep='\t',nrows=numberofrows,header=0)
7  title_principals = pd.read_csv("/content/drive/MyDrive/Dataset/Original/
       title_principals.tsv",nrows=numberofrows,sep='\t',header=0)
8
9  keep_col = ["tconst","titleType","primaryTitle","originalTitle","startYear",
       "runtimeMinutes","genres"]
10 title_basics = title_basics[keep_col]
11 title_basics = title_basics[title_basics["titleType"].str.contains("movie")
       == True]
12
13 print(title_basics.head(3))
14
15 merged1=pd.merge(title_basics,title_principals,how='inner',on='tconst')
16 del title_basics,title_principals
17 print(merged1)
18
19 name_basics=pd.read_csv("/content/drive/MyDrive/Dataset/Original/name_basics
       .tsv",sep='\t',nrows=numberofrows,header=0)
20
21 merged2=pd.merge(merged1,name_basics,how='inner',on='nconst')
22 del merged1
23 merged2=merged2.drop(columns=["ordering","nconst","birthYear","deathYear","
       knownForTitles","primaryProfession"])
24 del name_basics
25 print(merged2)
26
27 category=merged2.groupby('tconst')['category'].apply(list).reset_index(name=
       'category')
28 job=merged2.groupby('tconst')['job'].apply(list).reset_index(name='job')
29 characters=merged2.groupby('tconst')['characters'].apply(list).reset_index(
       name='characters')
30 primaryName=merged2.groupby('tconst')['primaryName'].apply(list).reset_index
       (name='primaryName')
31 result=merged2.drop_duplicates(subset=['tconst'])
32 result=result.drop(['category'], axis=1).drop(['job'], axis=1).drop(['
       characters'], axis=1).drop(['primaryName'], axis=1)
33 result=result.merge(category,on='tconst').merge(job,on='tconst').merge(
       characters,on='tconst').merge(primaryName,on='tconst')
34 print(result)
35
36 result.to_csv("/content/drive/MyDrive/Dataset/resultSetFinale.csv",index=
       False)
```

A .2 Creation of one single dataset from the csv kaggle file

```
1
2  import pandas as pd
3  from google.colab import drive
4  drive.mount('/content/drive')
```

```
numberofrows=None #100000
movies = pd.read_csv("/content/drive/MyDrive/Dataset/Original/rotten_movies.csv",nrows=numberofrows,header=0)
reviews = pd.read_csv("/content/drive/MyDrive/Dataset/Original/rotten_reviews.csv",nrows=numberofrows,header=0)
to_keep = ["rotten_tomatoes_link", "movie_title", "production_company","critics_consensus",
           "tomatometer_status", "tomatometer_rating", "tomatometer_count",
           "audience_status", "audience_rating", "audience_count",
           "tomatometer_top_critics_count", "tomatometer_fresh_critics_count",
           "tomatometer_rotten_critics_count"]

movies = movies[to_keep]
to_drop = ["publisher_name"]
reviews=reviews.drop(columns=to_drop)

merged=pd.merge(movies,reviews,how='inner',on="rotten_tomatoes_link")
print(merged)

categories = {}
arr = ["critic_name", "top_critic", "review_type", "review_score", "review_date", "review_content"]
for x in arr:
  categories[x]=merged.groupby('rotten_tomatoes_link')[x].apply(list).reset_index(name=x)

result=merged.drop_duplicates(subset=['rotten_tomatoes_link'])
for x in arr:
  result=result.drop([x], axis=1)
for x in arr:
  result=result.merge(categories[x],on='rotten_tomatoes_link')

print(result)

result.to_csv("/content/drive/MyDrive/Dataset/resultSetRotten.csv",index=False)
```

A .3 Merging of the file generated in the previous script

```

import pandas as pd
from google.colab import drive
drive.mount('/content/drive')
numberofrows=None
imdb = pd.read_csv("/content/drive/MyDrive/Dataset/resultSetFinale.csv",nrows=numberofrows,header=0)
rotten = pd.read_csv("/content/drive/MyDrive/Dataset/resultSetRotten.csv",nrows=numberofrows,header=0)

merged = {}
choose = ['primaryTitle', 'originalTitle']
rowHeadDataset = 20
for x in choose:
  merged[x]=pd.merge(imdb,rotten,how='inner', left_on=x, right_on='movie_title')
  print(x)
  merged[x]=merged[x].drop_duplicates(subset=[x])
  merged[x]=merged[x].drop(columns=['titleType','tomatometer_count','tomatometer_top_critics_count'])
  merged[x]=merged[x].rename(columns={'startYear':'year'})
```

```
  merged[x]=merged[x].drop(columns=['rotten_tomatoes_link', 'movie_title']+[
    j for j in choose if j!=x])
  print(len(pd.unique(merged[x][x])))
  print(list(merged[x]))
  print("====================")
  merged[x].to_csv(f"/content/drive/MyDrive/Dataset/ImdbJoinRotten{x}.csv",
    index=False)
  merged[x]=merged[x].head(rowHeadDataset)
  merged[x].to_csv(f"/content/drive/MyDrive/Dataset/headDataset{x}.csv",
    index=False)
```

A .4 Collapsing different rows in a single one generating an array for personnel field

```

import pandas as pd
from ast import literal_eval
from google.colab import drive
drive.mount('/content/drive')

numberofrows=None
df = pd.read_csv("/content/drive/MyDrive/Dataset/ImdbJoinRottenprimaryTitle.
    csv",nrows=numberofrows,header=0)

#print([x.split(',') for x in df['genres']])
print(df)

col = ["primaryName","category","job","characters"]
col1 = ["critic_name","top_critic","review_type","review_score","review_date
    ", "review_content"]

df['personnel'] = ""
df['review'] = ""

for row in range(df[col[0]].size):
  it = df['genres'][row]
  df['genres'][row] = ['"' + x + '"' for x in it.split(',')] if it != '\\N'
    else []
  tmp = []
  for c in col:
    tmp.append({c:eval(df[c][row])})
  res = []
  for c in range(len(tmp[0][col[0]])):
    res.append({})
  for i, j in zip(col, tmp):
    for idx, x in enumerate(j[i]):
      #print(i, idx, x)
      if x != '\\N':
        if i == 'characters':
          x = eval(x)
        res[idx]["'" + i + "'"] = "'" + str(x).replace("'", "##single-quote
    ##").replace('"', "##double-quote##") + "'"
  df['personnel'][row] = list(res)
  #print(res)
  ###
  tmp = []
  for c in col1:
    to_eval = df[c][row].replace('nan', 'None')
    arr = eval(to_eval)
    if c == "review_date":
```

```
      for i, elem in enumerate(arr):
        arr[i] = elem + "T00:00:00.000+00:00"
    tmp.append({c:arr})
    #print(tmp)
  res = []
  for c in range(len(tmp[0][col1[0]])):
    res.append({})
  for i, j in zip(col1, tmp):
    for idx, x in enumerate(j[i]):
      #print(i, idx, x)
      if x != '\\N':
        res[idx]["'" + i + "'"] = "'" + str(x).replace("True", "true").
   replace("False", "false").replace("'", "##single-quote##").replace('"', "
   ##double-quote##") + "'"
  df['review'][row] = list(res)
  #df['review'][row] = eval(str(res))
  #print(res)
  #print()

df=df.drop(columns=col)
df=df.drop(columns=col1)
df=df.drop(columns=['tconst'])

print(df["review"][0])

it = df['personnel'][0]#[4]['review_content']
print(type(it))
print(it)

df.to_csv("/content/drive/MyDrive/Dataset/
   movieCollectionEmbeddedReviewPersonnel.csv",index=False)
df = df.head(20)
df.to_csv("/content/drive/MyDrive/Dataset/
   headmovieCollectionEmbeddedReviewPersonnel.csv",index=False)
```

A .5 Generates a hashed password for all the users

```
import hashlib
#from pprint import pprint as print
from pymongo import MongoClient

def get_database():
   CONNECTION_STRING = "mongodb://localhost:27017"
   client = MongoClient(CONNECTION_STRING)
   return client['rottenMovies']

if __name__ == "__main__":
    dbname = get_database()
    collection = dbname['user']
    total = collection.count_documents({})
    for i, user in enumerate(collection.find()):
        all_reviews = user['last_3_reviews']
        sorted_list = sorted(all_reviews, key=lambda t: t['review_date'])
   [-3:]

        hashed = hashlib.md5(user["username"].encode()).hexdigest()

        newvalues = { "$set": { 'password': hashed, 'last_3_reviews':
   sorted_list } }
        filter = { 'username': user['username']}
        collection.update_one(filter, newvalues)
```

```
        print(f"{i/total:%}\r", end='')
    print()
```

A .6 Generates the graph database by randomly which user follows a top critic

```
from pymongo import MongoClient
from neo4j import GraphDatabase
from random import randint, shuffle

def get_database():
   CONNECTION_STRING = "mongodb://localhost:27017"
   client = MongoClient(CONNECTION_STRING)
   return client['rottenMovies']

class Neo4jGraph:

    def __init__(self, uri, user, password):
        self.driver = GraphDatabase.driver(uri, auth=(user, password),
   database="rottenmoviesgraphdb")

    def close(self):
        self.driver.close()

    def addUser(self, uid, name, isTop):
        with self.driver.session() as session:
            if isTop:
                result = session.execute_write(self._addTopCritic, uid, name
   )
            else:
                result = session.execute_write(self._addUser, uid, name)

    def addMovie(self, mid, title):
        with self.driver.session() as session:
            result = session.execute_write(self._addMovie, mid, title)

    def addReview(self, name, mid, freshness, content, date):
        with self.driver.session() as session:
            result = session.execute_write(self._addReview, name, mid,
   freshness, content, date)

    def addFollow(self, uid, cid):
        with self.driver.session() as session:
            result = session.execute_write(self._addFollow, uid, cid)

    @staticmethod
    def _addUser(tx, uid, name):
        query = "CREATE (n:User{id:\"" + str(uid) + "\", name:\"" + name.
   replace('"', '\\"') + "\"})"
        #print(query)
        result = tx.run(query)

    @staticmethod
    def _addTopCritic(tx, cid, name):
        query = "CREATE(m:TopCritic{id:\"" + str(cid) + "\", name:\"" + name
   .replace('"', '\\"') + "\"})"
        #print(query)
        result = tx.run(query)

    @staticmethod
```

```
50      def _addMovie(tx, mid, title):
51          query = "CREATE(o:Movie{id:\"" + str(mid) + "\", title:\"" + title.
    replace('"', '\\"') + "\"})"
52          #print(query)
53          result = tx.run(query)
54
55      @staticmethod
56      def _addReview(tx, name, mid, freshness, content, date): # date in
    format YYYY-mm-dd, freshness in [TRUE, FALSE]
57          query = "MATCH(n{name:\"" + str(name).replace('"', '\\"') + "\"}), (
    m:Movie{id:\"" + str(mid) + "\"}) CREATE (n)-[r:REVIEWED{freshness:" +
    freshness + ", date:date('" + date + "'), content:\"" + content.replace('
    "', '\\"') + "\"}]->(m)"
58          #print(query)
59          result = tx.run(query)
60
61      @staticmethod
62      def _addFollow(tx, uid, cid):
63          query = "MATCH(n:User{id:\"" + str(uid) + "\"}), (m:TopCritic{id:\""
     + str(cid) + "\"}) CREATE (n)-[r:FOLLOWS]->(m)"
64          #print(query)
65          result = tx.run(query)
66
67  if __name__ == "__main__":
68      # dbs initialization
69      dbname = get_database()
70      graphDB = Neo4jGraph("bolt://localhost:7687", "neo4j", "password")
71
72      # user creation
73      collection = dbname['user']
74      total = collection.count_documents({})
75      print(f"user {total = }")
76      for i, user in enumerate(list(collection.find({}, {"_id":1, "username"
    :1, "date_of_birth":1}))):
77          graphDB.addUser(user['_id'], user['username'], 'date_of_birth' not
    in user)
78          if not i%100:
79              print(f"{(i+1)/total:%}\r", end='')
80
81      # movie creation and review linking
82      collection = dbname['movie']
83      total = collection.count_documents({})
84      print(f"\nmovie {total = }")
85      for i, movie in enumerate(list(collection.find({}, {"_id":1, "
    primaryTitle":1, "review":1}))):
86          graphDB.addMovie(movie['_id'], movie['primaryTitle'])
87          movie['review'] = list({v['critic_name']:v for v in movie['review'
    ]}.values()) # make unique reviews per critic
88          for rev in movie['review']:
89              graphDB.addReview(rev['critic_name'], movie['_id'], {"Fresh":"
    TRUE", "Rotten":"FALSE"}[rev['review_type']], str(rev['review_content'])
    [:15], str(rev['review_date'])[:10])
90          print(f"{(i+1)/total:%}\r", end='')
91
92      # follow linking
93      collection = dbname['user']
94      uids = [x['_id'] for x in list(collection.find({"date_of_birth":{"
    $exists":True}}, {"_id":1}))]
95      cids = [x['_id'] for x in list(collection.find({"date_of_birth":{"
    $exists":False}}, {"_id":1}))]
96      total = len(uids)
```

```
    print(f"\nfollow {total = }")
    for i, user in enumerate(uids):
        shuffle(cids)
        for j in range(randint(0, 20)):
            graphDB.addFollow(user, cids[j])
        print(f"{i/total:%}\r", end='')

    graphDB.close()
```

B Mongosh scripts

B .1 Modify the db fields to fit the decided structure

```

db.movie.find().forEach(
    x => {
        print(x.primaryTitle);
        x.review = JSON.parse(
            x.review.replaceAll('"\'', '"')
                .replaceAll('\'"', '"')
                .replaceAll('"false"', 'false')
                .replaceAll('"true"', 'true')
                .replaceAll('"None"', 'null')
                .replaceAll(/\\x\d{2}/g, "")
                .replaceAll("##single-quote##", "\'")
                .replaceAll("##double-quote##", '\\"')
                .replaceAll("\\x", "x")
        );
        x.personnel = JSON.parse(
            x.personnel.replaceAll('"\'', '"')
                .replaceAll('\'"', '"')
                .replaceAll('"None"', 'null')
                .replaceAll("##single-quote##", '\'')
                .replaceAll("##double-quote##", '\\"')
                .replaceAll('"[\'', '["')
                .replaceAll('"[\\"', '["')
                .replaceAll('\']"', '"]')
                .replaceAll('\\"]"', '"]')
                .replaceAll(/(\[[^[:]*)\\", \\"([^]:]*\])/g, '$1", "$2')
                .replaceAll(/(\[[^[:]*)\', \\"([^]:]*\])/g, '$1", "$2')
                .replaceAll(/(\[[^[:]*)\\", \'([^]:]*\])/g, '$1", "$2')
        );
        x.genres = JSON.parse(
            x.genres = x.genres.replaceAll('"\'', '"')
                    .replaceAll('\'"', '"')
                    .replaceAll('"None"', 'null')
                    .replaceAll("##single-quote##", "\'")
                    .replaceAll("##double-quote##", '\\"')
        );
        db.movie.updateOne(
            {"_id": x._id},
            {$set:
                {
                    "review": x.review,
                    "personnel": x.personnel,
                    "genres": x.genres,
                    "runtimeMinutes":parseInt(x.runtimeMinutes),
                    "year":parseInt(x.year),
                    "tomatometer_rating":parseFloat(x.tomatometer_rating),
                    "audience_rating":parseFloat(x.audience_rating),
```

```
                    "audience_count":parseFloat(x.audience_count),
                    "tomatometer_fresh_critics_count":parseInt(x.
    tomatometer_fresh_critics_count),
                    "tomatometer_rotten_critics_count":parseInt(x.
    tomatometer_rotten_critics_count)
                }
            }
        );
    }
);
```

B .2 Modify the db field by removing removing audience count

```
total = db.movie.find().count();
i = 0;
db.movie.find().forEach(
    x => {
        print(x.primaryTitle);
        x.review.forEach(rev =>{
            if(typeof (rev.review_date) === "string" ){
                db.movie.updateOne(
                    {primaryTitle: x.primaryTitle },
                    { $set: { "review.$[elem].review_date" : new Date(rev.
    review_date) } },
                    { arrayFilters: [ { "elem.critic_name": rev.critic_name
    } ] }
                 )
            }
        })
        print(100*i++/total);
});
```

B .3 Create a new collection for the user based on the data present in the movie collection

```
total = db.runCommand({ distinct: "movie", key: "review.critic_name", query:
    {"review.critic_name":{$ne:null}}}).values.length
i = 0;
db.runCommand(
{ distinct: "movie", key: "review.critic_name", query: {"review.critic_name"
    :{$ne:null}}}).values.forEach(
    (x) => {
        review_arr = []
        movie_arr = []
        is_top = false
        db.movie.aggregate(
            [
                { $project:
                    {
                        index: { $indexOfArray: ["$review.critic_name", x]},
                        primaryTitle: 1
                    }},
                {$match:{index:{$gt:-1}}}
            ]
        ).forEach(
            y => {
                tmp = db.movie.aggregate([
                    {
                        $project:
                        {
```

```
                            top_critic: {
                                $arrayElemAt: ["$review.top_critic", y.index
    ]
                            },
                            primaryTitle: y.primaryTitle,
                            review_type: {
                                $arrayElemAt: ["$review.review_type", y.
    index]
                            },
                            review_score: {
                                $arrayElemAt: ["$review.review_score", y.
    index]
                            },
                            review_date: {
                                $arrayElemAt: ["$review.review_date", y.
    index]
                            },
                            review_content: {
                                $arrayElemAt: ["$review.review_content", y.
    index]
                            }
                        }
                    },
                    {
                        $match:{_id:{$eq:y._id}}
                    }
                ]).toArray()[0];
                is_top |= tmp.top_critic;
                review_arr.push(tmp)
                //movie_arr.push(tmp._id)
                movie_arr.push({"movie_id": tmp._id, "primaryTitle": y.
    primaryTitle, "review_index": y.index})
            })

        name_parts = x.split(/\s/)
        first_name = name_parts.splice(0, 1)[0]
        last_name = name_parts.join(' ')

        print(100*i++/total, x, is_top)
        //print(first_name, ':', last_name)
        //print(review_arr)
        //print(movie_arr)
        db.user.insertOne(
            {
                "username": x,
                "password": "",
                "first_name": first_name,
                "last_name": last_name,
                "registration_date": new Date("2000-01-01"),
                "last_3_reviews": review_arr,
                "reviews" : movie_arr
            }
        );
        if (!is_top){
            db.user.updateOne(
                {"username": x},
                {$set:
                    {"date_of_birth": new Date("1970-07-20")}
                }
            )
        }
```

```
79          print("================================")
80      }
81  )
```

C matematica

$$x^2 - 5x + 6 = 0 \tag{2.1}$$