به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

درس کلان داده

تمرین شماره ۲

?

اردیبهشت ماه ۱۳۹۹

فهرست

بخش اول

پیکر بندی

در این بخش به پیکربندی MongoDB می‌پردازیم. با توجه به قدیمی بودن نسخه MongoDB موجود در repository رسمی Ubuntu 18.04، لازم است تا آخرین نسخه MongoDB از مخزن رسمی MongoDB دریافت گردد. بدین منظور لازم است تا دستورات زیر در bash اجرا شود:

```
$ wget -qO - https://www.mongodb.org/static/pgp/server-4.2.asc | sudo apt-key add -
$ echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/4.2 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-4.2.list
$ sudo apt update
$ sudo apt-get install -y mongodb-org
$ cd ~
$ sudo mkdir -p data/db
$ sudo mongod --dbpath ~/data/db
```

پس از انجام این مراحل، آخرین نسخه از MongoDB نصب شده و دیتای آن نیز در دایرکتوری تعریف شده در مسیر کاربر فعلی ذخیره خواهد شد.

گام اول

پس از پیکربندی MongoDB، ابتدا اطلاعات چند کاربر محدود از API سایت randomuser دریافت شده است. برای وارد کردن این اطلاعات به صورت دستی در دیتابیس، از دستورات mongo shell کمک می‌گیریم. خط فرمان mongo shell با وارد کردن دستور زیر باز خواهد شد:

```
$ mongo
```

که پس از آن می‌توان دستورات را خط به خط درون آن وارد کرد. اما این امکان نیز وجود دارد که دستورات mongo shell را در یک فایل اسکریپت درج کرده و همگی را به صورت یکجا اجرا کنیم. این کار می‌تواند توسط اجرای دستور زیر صورت گیرد:

```
$ mongo < step1.js
```

که فایل step1.js شامل دستورات mongo shell بوده و همگی در این مرحله توسط mongo shell اجرا می‌گردند. در این اسکریپ با استفاده از تابع insertOne یک document به یک collection اضافه شده است. همچنین با استفاده از تابع count تعداد documentهای موجود در یک collection قابل بررسی است.

```
use bigdata99
print("Deleting all data in tempusers...")
db.tempusers.deleteMany({})
print("Adding 3 sample data to tempusers...")
db.tempusers.insertOne({
      …
})
db.tempusers.insertOne({
      …
})
db.tempusers.insertOne({
      …
})
print("Count of tempusers: ")
db.tempusers.count()
print("One record of tempusers: ")
db.tempusers.findOne()
```

فایل **step1.js**

در مرحله بعد این فرآیند را با استفاده از پایتون و کتابخانه رسمی pymongo انجام می‌دهیم. در این مرحله با اتصال به درگاه randomuser در هر مرحله اقدام به دریافت اطلاعات ۵ هزار کاربر خواهیم کرد.

اسکریپت مربوط به این بخش در فایل step1.py موجود است. این اسکریپت در هر بار اجرا تا زمانی که درگاه randomuser اطلاعات کاربران را بدست بدهد، اقدام به دریافت داده‌ها در دسته‌های ۵ هزارتایی کرده، سپس آن‌ها را در دیتابیس MongoDB وارد کرده و در ادامه بررسی می‌کند که تا این لحظه چند کاربر در دیتابیس موجود می‌باشند. در صورتی که این تعداد از ۱۰۰ هزار کمتر باشد، مراحل را از ابتدا دنبال می‌کند. در صورتی که این تعداد به ۱۰۰ هزار رسید، کار دریافت دیتا را متوقف کرده و موفقیت آمیز بودن فرآیند را اعلام می‌کند.

با توجه به اینکه پس از تعداد محدود دریافت دسته‌های ۵ هزارتایی کاربر از randomuser، به مدت چند دقیقه پاسخگویی از سمت این API صورت نمی‌گیرد، برای تسریع در فرآیند دریافت دیتا در صورت از بین رفتن اطلاعات داخل دیتابیس، اسکریپت به صورت خودکار ۱۰۰ هزار داده دیتابیس را در صورت کامل بودن تعداد آن‌ها در یک فایل tar.gz. که فشرده شده فایل json داده‌های دریافت شده است به عنوان Backup ذخیره خواهد کرد. این اطلاعات در زمان لازم توسط فایل import.py می‌توانند مجدد به دیتابیس MongoDB وارد شوند.

```
#!/usr/bin/python3
import requests
import json
import time
from pymongo import MongoClient
import tarfile
import os
client = MongoClient()
db = client['bigdata99']
url = 'https://randomuser.me/api/'
api_sleep = 10
api_limit = 5000
total = 100000
#db.users.remove({})
fetched = db.users.count()
while fetched < total:
    response = requests.request('GET', url, params={'nat':'ir',
    'results':api_limit})
    if response.status_code == requests.codes.ok:
       data = response.json()['results']
       try:
              db.users.insert_many(data)
       except Exception as e:
              print("Mongo insert exception: "+str(e))
       fetched = db.users.count()
    else:
       print("Response code error: "+str(response.status_code))
       print("Trying again")
    print(f'Fetched and inserted {fetched} users so far')
    print(f'Waiting for {api_sleep} seconds...')
    time.sleep(api_sleep)
fetched = db.users.count()
print(f'Successfully fetched and inserted {fetched} users into the db.')
with open('users.json', 'w') as file:
    file.write('[')
    for document in db.users.find({}, {'_id': False}):
       file.write(json.dumps(document))
       file.write(',')
    file.write(']')
with tarfile.open('users.tar.gz','w:gz') as tar:
    tar.add('users.json')
os.remove('users.json')
```

فایل **step1.py**

در طی این فرآیند، با استفاده از تابع insertMany از MongoDB امکان وارد کردن دیتای چند کاربر به دیتابیس ممکن خواهد بود. همچنین صدا زدن تابع count روی یک collection تعداد documentهای موجود در آن دیتابیس را بدست می‌دهد و ازین طریق می‌توان اطمینان حاصل کرد که رکوردها در دیتابیس ذخیره شده اند یا خیر.

گام دوم

۱

فایل کد: step2-1.py

```
#!/usr/bin/python3

import pprint
from pymongo import MongoClient

client = MongoClient()
db = client['bigdata99']
cursor_limit = 10

users = db.users.find({ "dob.age": { "$gt": 50 }, "location.city":
    "نیشابور" }, {"name.first": True, "name.last": True, "dob.age":
    True, "location.city": True, "_id": False}).limit(cursor_limit)
print(f'Found {users.count()} results. Here are {cursor_limit}
    samples:')
for user in users:
    pprint.pprint(user)
```

زمان اجرا: ۰٫۶۶۱ ثانیه

تعداد خروجی: ۹۲۳ سند

نمونه خروجی:

```
Found 923 results. Here are 10 samples:
{'dob': {'age': 67},
 'location': {'city': 'نیشابور'},
 'name': {'first': 'مریم', 'last': 'قاسمی'}}
{'dob': {'age': 65},
 'location': {'city': 'نیشابور'},
 'name': {'first': 'آتنا', 'last': 'یزدان'}}
{'dob': {'age': 73},
 'location': {'city': 'نیشابور'},
 'name': {'first': 'مریم', 'last': 'احمدی'}}
{'dob': {'age': 58},
 'location': {'city': 'نیشابور'},
 'name': {'first': 'بهاره', 'last': 'سهیلی دار'}}
{'dob': {'age': 63},
 'location': {'city': 'نیشابور'},
 'name': {'first': 'طاهره', 'last': 'عیسی'}}
{'dob': {'age': 58},
 'location': {'city': 'نیشابور'},
 'name': {'first': 'علی', 'last': 'پاسرا'}}
{'dob': {'age': 52},
 'location': {'city': 'نیشابور'},
 'name': {'first': 'مریم', 'last': 'رضایی'}}
{'dob': {'age': 67},
 'location': {'city': 'نیشابور'},
 'name': {'first': 'سارا', 'last': 'یزدی'}}
{'dob': {'age': 75},
 'location': {'city': 'نیشابور'},
 'name': {'first': 'رضا', 'last': 'اورماک'}}
{'dob': {'age': 60},
 'location': {'city': 'نیشابور'},
 'name': {'first': 'علی', 'last': 'مرادخانی'}}
```

۲

فایل کد: step2-2.py

```
#!/usr/bin/python3

import pprint
from pymongo import MongoClient

client = MongoClient()
db = client['bigdata99']
cursor_limit = 10

users = db.users.find({ "registered.age": { "$gt": 20 } },
    {"name.last": True, "location": True, "phone": True,
    "registered.age": True, "_id": False}).limit(cursor_limit)
print(f'Found {users.count()} results. Here are {cursor_limit}
    samples:')
for user in users:
    pprint.pprint(user)
```

زمان اجرا: ۰٫۶۴۹ ثانیه

تعداد خروجی: ۰ سند

نمونه خروجی:

```
Found 0 results. Here are 10 samples:
```

۳

فایل کد: step2-3.py

```
#!/usr/bin/python3

import pprint
from pymongo import MongoClient

client = MongoClient()
db = client['bigdata99']
cursor_limit = 10

db.users.update_many({},[{
    "$set": {
       "dob.year_persian": {
              "$cond": [
                     {
                            "$and": [
                                   { "$gte": [{ "$month": {"$dateFromString":
    {"dateString": "$dob.date"}} }, 3] },
                                   { "$gte": [{ "$dayOfMonth":
    {"$dateFromString": {"dateString": "$dob.date"}} }, 20] },
                            ]
                     },
                     {"$subtract": [{ "$year": {"$dateFromString":
    {"dateString": "$dob.date"}} }, 621]},
                     {"$subtract": [{ "$year": {"$dateFromString":
    {"dateString": "$dob.date"}} }, 622]}
              ]
       },
       "registered.year_persian": {
              "$cond": [
                     {
                            "$and": [
                                   { "$gte": [{ "$month": {"$dateFromString":
    {"dateString": "$registered.date"}} }, 3] },
                                   { "$gte": [{ "$dayOfMonth":
    {"$dateFromString": {"dateString": "$registered.date"}} }, 20] },
                            ]
                     },
                     {"$subtract": [{ "$year": {"$dateFromString":
    {"dateString": "$registered.date"}} }, 621]},
                     {"$subtract": [{ "$year": {"$dateFromString":
    {"dateString": "$registered.date"}} }, 622]}
              ]
       },
    }
}])


users = db.users.find().limit(cursor_limit)
print(f'Found {users.count()} results. Here are {cursor_limit} samples:')
for user in users:
    pprint.pprint(user)
```

زمان اجرا: ۷٫۶۷۵ ثانیه

نمونه خروجی:


```
{'_id': ObjectId('5ebbe8afcb04812d3f977cd3'),
 'cell': '0972-688-2403',
 'dob': {'age': 46, 'date': '1974-06-08T08:05:47.052Z', 'year_persian': 1352},
 'email': 'ln.njty@example.com',
 'gender': 'female',
 'id': {'name': '', 'value': None},
 'location': {'city': 'دزی',
              'coordinates': {'latitude': '-3.1461', 'longitude': '48.0007'},
              'country': 'Iran',
              'postcode': 74197,
              'state': 'ن انفسلگ',
              'street': {'name': 'ی بونج دربن', 'number': 9822},
              'timezone': {'description': 'Tehran', 'offset': '+3:30'}},
 'login': {'md5': '8bf12f6f160202bf61fe73c8a36e7a99',
           'password': 'april1',
           'salt': 'WOFBjs56',
           'sha1': 'b1a1e71e1c1911810372cc78faf06333df702229',
           'sha256': '48428de72f6706bc6d08282586f5bb8546a0a57c0108ec19d6f4032d1e6bef17',
           'username': 'smallkoala365',
           'uuid': '2b69fdfe-bca8-4724-ae1c-64812b32712e'},
 'name': {'first': 'انل ا', 'last': 'ی تاجن', 'title': 'Mrs'},
 'nat': 'IR',
 'phone': '072-14019832',
 'picture': {'large': 'https://randomuser.me/api/portraits/women/13.jpg',
             'medium': 'https://randomuser.me/api/portraits/med/women/13.jpg',
             'thumbnail': 'https://randomuser.me/api/portraits/thumb/women/13.jpg'},
 'registered': {'age': 17,
                'date': '2003-04-27T05:27:41.444Z',
                'year_persian': 1382}}
```


توضیحات:

در این بخش با الحاق aggregation pipeline در دستور updateMany اقدام به بررسی تاریخ‌های تولید و ثبت نام کاربران گردید. الگوریتم ایجاد سال شمسی از تاریخ میلادی به این صورت اعمال شده است که اگر تاریخ پس از روز ۲۰ ام ماه ۳ میلادی بود، از سال میلادی عدد ۶۲۱، و الا عدد ۶۲۲ کم گردد.

۴

فایل کد: step2-4.py

```
#!/usr/bin/python3

import pprint
import itertools
from pymongo import MongoClient

client = MongoClient()
db = client['bigdata99']
limit = 3

users = db.users.aggregate([
    { "$match":
       { "$expr":
              { "$and": [
                     { "$eq": [
                            { "$month": {"$dateFromString": {"dateString":
    "$dob.date"}} },
                            { "$month": "$$NOW" }
                     ]},
                     { "$eq": [
                            { "$dayOfMonth": {"$dateFromString": {"dateString":
    "$dob.date"}} },
                            { "$dayOfMonth": "$$NOW" }
                     ]}
              ]}
       }
    },
    {
       "$project": {
              "name.first": True,
              "name.last": True,
              "email": True,
              "dob.date": True,
              "_id": False
       }
    }
])


users = list(users)
print(f'Found {len(users)} results. Here are {limit} samples:')
for user in users[:limit]:
    pprint.pprint(user)c
```

زمان اجرا: ۱,۶۵۳ ثانیه

تعداد خروجی: ۲۵۴ سند (تاریخ ۲۰۲۰/۵/۱۹)

نمونه خروجی:

```
Found 254 results. Here are 3 samples:
{'dob': {'date': '1987-05-19T04:29:25.272Z'},
 'email': 'pwry.rdyy@example.com',
 'name': {'first': 'ای‌روپ', 'last': 'ی‌اضر'}}
{'dob': {'date': '1946-05-19T01:18:20.052Z'},
 'email': 'ard.sdr@example.com',
 'name': {'first': 'دارآ', 'last': 'ردص'}}
{'dob': {'date': '1998-05-19T00:50:32.068Z'},
 'email': 'aasl.khmrw@example.com',
 'name': {'first': 'لسع', 'last': 'اورم‌اک'}}
```

۵

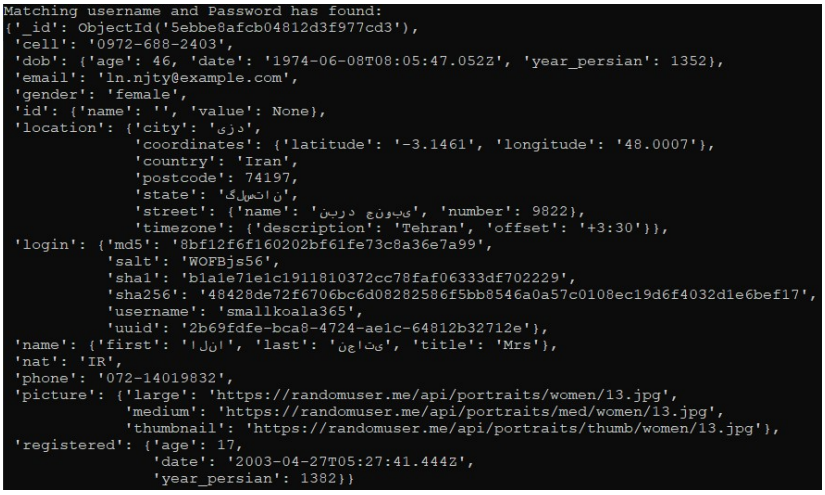
فایل کد: step2-5.py

```
#!/usr/bin/python3

import pprint
import itertools
from hashlib import sha256
from pymongo import MongoClient

client = MongoClient()
db = client['bigdata99']
limit = 3

user_name = "smallkoala365"
password = "april1"

users = db.users.find({"login.username": user_name},{"login.password":
    False})
found = False
for user in users:
    salt = user["login"]["salt"]
    if user["login"]["sha256"] == sha256((password+salt).encode('utf-
    8')).hexdigest():
       print("Matching username and Password has found:")
       pprint.pprint(user)
       found = True
if found == False:
    print("No matching username and password has found.")
```

زمان اجرا: ۰,۶۵۶ ثانیه

تعداد خروجی: ۱ سند (نام کاربری smallkoala365 و رمزعبور april1)

نمونه خروجی:



توضیحات:

همانطور که در صورت سؤال نیز ذکر شده است، ذخیره رمز عبور در دیتابیس به صورت خام کاری اشتباه است، زیرا که اگر یک حمله کننده به هر نحوی به اطلاعات دیتابیس دسترسی پیدا کند، رمز عبور تمامی کاربران افشاء خواهد شد. برای رفع این مشکل می‌توان به جای خود رمزعبور، تابع Hash را بر آن‌ها اعمال کرده و سپس نتیجه را در دیتابیس ذخیره نمود. تابع Hash خصوصیتی که دارد، یکطرفه بودن آن و عدم امکان یافتن معکوس آن می‌باشد. همچنین تابع Hash ویژگی‌های دیگری نظیر collision resistance و غیره را نیز دارد که توضیح آن‌ها در مجال این گزارش نیست.

توابع Hash مختلفی نظیر SHA1، SHA256 و یا MD4 وجود دارد که می‌توان از آن‌ها بهره جست. امن ترین این توابع SHA256 می‌باشد.

نکته دیگر در این مرحله اینست که اعمال تابع Hash به تنهایی نیز ممکن است امنیت را تأمین نکند، زیرا که حمله کننده می‌تواند از پیش یک دیتاست شامل کلمه عبورهای مختلف و Hash آن‌ها را تدارک دیده باشد و پس از دسترسی به دیتابیس، با یک عملیات Lookup ساده در جدول خود (Rainbow Table) به یافتن مقادیر معکوس Hash اقدام کند. برای جلوگیری از این کار، به ازای هر کاربر یک مقدار تصادفی با نام Salt به ابتدا و یا انتهای رمز عبور کاربر Concate شده و سپس از تمامی آن Hash گرفته می‌شود. همچنین مقدار Salt به صورت خام نیز در کنار Hash گرفته شده در دیتابیس ذخیره می‌گردد. حال در صورتی که یک حمله کننده به دیتابیس دسترسی پیدا کند، با توجه به اینکه رمز عبور هر کاربر با یک Salt نیز الحاق شده است، به ازای هر کاربر می‌بایست یک Rainbow Table تشکیل داده که این مدت زمان انجام عملیات را غیر قابل دسترس می‌نماید.

در دیتاست فعلی مقادیر SHA256 و یک Salt تصادفی ذخیره شده است. با بررسی این دیتاست، مشخص می‌شود که مقدار Hash ذخیره شده برای هر کاربر برابر Concate مقدار Salt به انتهای رمز عبور کاربر بوده که تابع Hash بر روی تمام آن اعمال شده است.

نکته قابل توجه اینست که متأسفانه مقادیر نام کاربری موجود برای کاربران یکتا نیست(!) و با یک مقدار نام کاربری می‌توان چندین کاربر را یافت که این مسئله غیر استاندارد بوده و امکان بررسی ورود یک کاربر را وابسته به مقدار رمز عبور وی نیز می‌نماید. با این حال کد مربوط به این بخش با عنایت به این مسئله طراحی شده است.

در فرآیند کار، می‌خواهیم یک کاربر با نام کاربری مشخص و رمز عبور مشخص را پیدا کنیم. ابتدا از دیتابیس تمامی کاربران با رمز عبور مشخص را می‌یابیم. سپس به ازای هر کاربر، مقدار Salt وی را به رمز

عبور مشخص Concate کرده و از مجموع آن‌ها مقدار SHA256 را می‌یابیم. سپس بررسی می‌کنیم که مقدار SHA256 ذخیره شده برای هر کاربر (که با فرمت HEX بوده) آیا با مقدار بدست آمده از SHA256 حاصل از Concate بدست آمده یکی هست یا خیر. اگر یکی بود، کاربر را به عنوان نتیجه اعلام می‌کنیم.

لازم به ذکر است که در این مرحله با توجه به وجود مقادیر Salt و SHA256 به ازای هر کاربر، نیازی به آپدیت دیتا درون دیتابیس نبوده و از قبل اینکار صورت گرفته است.

گام سوم

۱

فایل کد: step3-1.py

```
#!/usr/bin/python3

import pprint
import itertools
from pymongo import MongoClient

client = MongoClient()
db = client['bigdata99']

results = db.users.aggregate([
    {
       "$project": {
              "ageGroup": {
                     "$cond": [
                            { "$lte": ["$dob.age", 16] },
                            "young" ,
                            {
                                   "$cond": [
                                          { "$lte": ["$dob.age", 30] },
                                          "adult",
                                          "old"
                                   ]
                            }
                     ]
              },
              "dob.age": True
       }
    },
    {
       "$group": {
              "_id": "$ageGroup",
              "count": {
                     "$sum": 1
              }
       }
    }
])


results = list(results)
for result in results:
    pprint.pprint(result)
```

زمان اجرا: ۰,۶۵۱ ثانیه

نمونه خروجی:

```
{'_id': 'adult', 'count': 16124}
{'_id': 'old', 'count': 83876}
```

توضیحات:

در دیتای دریافت شده، هیچ رکوردی دارای سن زیر ۱۶ سال نبوده است.

۲

فایل کد: step3-2.py

```
#!/usr/bin/python3

import pprint
import itertools
from pymongo import MongoClient

client = MongoClient()
db = client['bigdata99']

results = db.users.aggregate([
    {
       "$group": {
              "_id": "$location.state",
              "count": {
                     "$sum": 1
              }
       }
    }
])


results = list(results)
for result in results:
    pprint.pprint(result)
```

زمان اجرا: ۰٫۶۵۴ ثانیه

نمونه خروجی:

```
{'_id': 'ن ان م س', 'count': 3291}
{'_id': 'ی وض ر ن ا س ا ر خ', 'count': 3304}
{'_id': 'ن ا ر د ن ز ا م', 'count': 3212}
{'_id': 'ق م', 'count': 3181}
{'_id': 'ن ا ل ی گ', 'count': 3135}
{'_id': 'ی ب ون چ ن ا س ا ر خ', 'count': 3343}
{'_id': 'ن ا ت س د ر ک', 'count': 3342}
{'_id': 'ی ر ا ی ت خ ب و ل ا ج م ر ا ه چ', 'count': 3214}
{'_id': 'س ر ا ف', 'count': 3200}
{'_id': 'ا ر د ب ی ل', 'count': 3202}
{'_id': 'ی ب ر غ ن ا چ ی ا ب ر د ز آ', 'count': 3225}
{'_id': 'ن ا ت س ل گ', 'count': 3149}
{'_id': 'ن ا ت س چ و ل ب و ن ا ت س ی س', 'count': 3261}
{'_id': 'ق ی ن ی و ز', 'count': 3262}
{'_id': 'ز ی د', 'count': 3278}
{'_id': 'ن ا ه ف ص ا', 'count': 3173}
{'_id': 'ن ا ت س ر ل', 'count': 3220}
{'_id': 'ی ل ا م ش ن ا س ا ر خ', 'count': 3199}
{'_id': 'ن ا گ ز م ر ه', 'count': 3262}
{'_id': 'ق ی ق ر ش ن ا چ ی ا ب ر د ز آ', 'count': 3198}
{'_id': 'د م ج ا ر ی و ب و ه ی و ل ی گ ه ک', 'count': 3175}
{'_id': 'ر ه ش و ب', 'count': 3230}
{'_id': 'ی ز ک ر م', 'count': 3249}
{'_id': 'ن ا ر ه ت', 'count': 3337}
{'_id': 'ن ا ت س ز و خ', 'count': 3230}
{'_id': 'ا ل ی ا م', 'count': 3218}
{'_id': 'ه ا ش ن ا م ر ک', 'count': 3140}
{'_id': 'ن ا د م ه', 'count': 3201}
{'_id': 'ز ر ی ب ل ا', 'count': 3121}
{'_id': 'ن ا م ر ک', 'count': 3242}
{'_id': 'ن ا چ ن ز', 'count': 3206}
```

۳

فایل کد: step3-3.py

```
#!/usr/bin/python3

import pprint
import itertools
from pymongo import MongoClient

client = MongoClient()
db = client['bigdata99']

db.users.update_many(
    { "location.timezone.offset": "+5:00" },
    { "$unset": {"cell": ""} }
)

results = db.users.aggregate([
    {
       "$group": {
              "_id": None,
              "count": {
                     "$sum": {
                            "$cond": [{"$ifNull": ["$cell", False]}, 0, 1]
                     }
              }
       }
    }
])

results = list(results)

for result in results:
    pprint.pprint(result)
```

زمان اجرا: ۰,۶۵۶ ثانیه

نمونه خروجی:

```
{'_id': None, 'count': 3490}
```

۴

فایل کد: step3-4.py

```
#!/usr/bin/python3

import pprint
import itertools
from pymongo import MongoClient

client = MongoClient()
db = client['bigdata99']

tehran_average = list(db.users.aggregate([
    {
       "$match": { "location.state": "تهران"}
    },
    {
       "$group": {
              "_id": None,
              "average": {
                     "$avg": "$dob.age"
              }
       }
    }
]))[0]["average"]

results = db.users.aggregate([
    {
       "$group": {
              "_id": "$location.state",
              "average": {
                     "$avg": "$dob.age"
              }
       }
    },
    {
       "$project": {
              "_id": True,
              "average": True,
              "difference": {"$subtract": ["$average", tehran_average]}
       }
    }
])


results = list(results)
for result in results:
    pprint.pprint(result)
```

زمان اجرا: ۰,۶۵۷ ثانیه

نمونه خروجی:


```
{'_id': 'دزی', 'average': 48.83343502135448, 'difference': -0.3370774149655631}
{'_id': 'نادفصا',
 'average': 48.81626221241727,
 'difference': -0.35425022390277405}
{'_id': 'نااحسرل',
 'average': 48.7416149068323,
 'difference': -0.42889752948774884}
{'_id': 'یلامش نااسارخ',
 'average': 48.8346358236949,
 'difference': -0.33587661262514246}
{'_id': 'نااگزمره',
 'average': 48.66339668914776,
 'difference': -0.5071157471722856}
{'_id': 'یقرش نااچیابرذآ',
 'average': 48.91400875547217,
 'difference': -0.2565036808478 78}
{'_id': 'دمحاریوب و هیولیگهک',
 'average': 49.333543307086615,
 'difference': 0.16303087076656908}
{'_id': 'رهشوب',
 'average': 48.83653250773994,
 'difference': -0.3339799285801064}
{'_id': 'یزکرم',
 'average': 49.08741151123422,
 'difference': -0.08310092508582301}
{'_id': 'نارهت', 'average': 49.170512436320045, 'difference': 0.0}
{'_id': 'نااحسزوخ',
 'average': 48.55634674922601,
 'difference': -0.6141656870940366}
{'_id': 'مالیا',
 'average': 48.873834679925416,
 'difference': -0.29667775639462945}
{'_id': 'هاشنامرک',
 'average': 49.15732484076433,
 'difference': -0.013187595555713472}
{'_id': 'نادمه',
 'average': 49.10996563573883,
 'difference': -0.0605468005812142}
{'_id': 'زربلا',
 'average': 48.35148990708107,
 'difference': -0.8190225292389783}
{'_id': 'نامرک',
 'average': 48.677359654534236,
 'difference': -0.4931527817858097}
{'_id': 'نااچنز',
 'average': 49.06363069245165,
 'difference': -0.10688174386839222}
```


توضیحات:

ابتدا میانگین برای استان تهران محاسبه شده و در مرحله بعد این اختلاف میانگین بین هر استان و تهران محاسبه شده است. همانطور که مشخص است، مقدار این اختلاف برای خود استان تهران برابر ۰ است.

۵

فایل کد: step3-5.py

```
#!/usr/bin/python3

import pprint
import itertools
from pymongo import MongoClient

client = MongoClient()
db = client['bigdata99']

max_province = list(db.users.aggregate([
    {
       "$group": {
              "_id": "$location.state",
              "count": {
                     "$sum": 1
              }
       }
    },
    {
       "$sort": { "count": -1 }
    },
    {
       "$limit": 1
    }
]))[0]

min_province = list(db.users.aggregate([
    {
       "$group": {
              "_id": "$location.state",
              "count": {
                     "$sum": 1
              }
       }
    },
    {
       "$sort": { "count": 1 }
    },
    {
       "$limit": 1
    }
]))[0]

print("Max Record:")
pprint.pprint(max_province)
print("Min Record:")
pprint.pprint(min_province)
```

زمان اجرا: ۰,۶۵۷ ثانیه

نمونه خروجی:

```
Max Record:
{'_id': 'یبونج ن اس ارخ', 'count': 3343}
Min Record:
{'_id': 'ا زربل', 'count': 3121}
```

گام چهارم

۲,۱

فایل کد: step4-2-1.py

```
#!/usr/bin/python3

import pprint
import itertools
from pymongo import MongoClient, DESCENDING

client = MongoClient()
db = client['bigdata99']

db.users.drop_indexes()

db.users.create_index([("dob.age", DESCENDING),("location.city",
    DESCENDING)])

for index in db.users.list_indexes():
    print(index)
```

مجموعه ایندکس پیشنهادی:

میزان بهبود عملکرد: ۰,۶۴۷-۰,۶۶۱

توضیحات: با توجه به حجم کم داده‌ها، عملکرد index در این حالت چشمگیر نمی‌باشد.

۲,۴

فایل کد: step4-2-4.py

```
#!/usr/bin/python3

import pprint
import itertools
from pymongo import MongoClient, DESCENDING

client = MongoClient()
db = client['bigdata99']

db.users.drop_indexes()

db.users.create_index([("dob.date", DESCENDING)])

for index in db.users.list_indexes():
    print(index)
```

مجموعه ایندکس پیشنهادی:

```
dob.date
```

میزان بهبود عملکرد: ۱,۶۵۰-۱,۶۵۵

توضیحات: در حقیقت این بخش استفاده از index امکان پذیر نیست و طبق انتظار تأثیری در زمان اجرا نداشته است. علت اینست که امکان تعریف index بر روی کل فیلد تاریخ درون دیتابیس ممکن بوده و امکان تعریف index تنها برای بخش ماه و سال تاریخ ممکن نمی‌باشد. برای اینکار پیشنهاد می‌شود که مقدار تاریخ به صورت یک آبجکت کلی تعریف نشده و مقادیری که امکان جستجو بر اساس آن‌ها در آینده بالاست، به صورت جداگانه در document ها تعریف گردند.

۲,۵

فایل کد: step4-2-5.py

```
#!/usr/bin/python3

import pprint
import itertools
from pymongo import MongoClient, DESCENDING

client = MongoClient()
db = client['bigdata99']

db.users.drop_indexes()

db.users.create_index([("login.username", DESCENDING)])

for index in db.users.list_indexes():
    print(index)
```

مجموعه ایندکس پیشنهادی:

```
login.username
```

میزان بهبود عملکرد: ۰,۶۴۵-۰,۶۵۶

توضیحات: با توجه به حجم کم داده‌ها، عملکرد index در این حالت چشمگیر نمی‌باشد.

۳,۲

فایل کد: step4-3-2.py

```
#!/usr/bin/python3

import pprint
import itertools
from pymongo import MongoClient, DESCENDING

client = MongoClient()
db = client['bigdata99']

db.users.drop_indexes()

db.users.create_index([("location.state", DESCENDING)])

for index in db.users.list_indexes():
    print(index)
```

مجموعه ایندکس پیشنهادی:

```
location.state
```

میزان بهبود عملکرد: ۰,۶۴۳-۰,۶۵۴

توضیحات: با توجه به حجم کم داده‌ها، عملکرد index در این حالت چشمگیر نمی‌باشد.

۳,۵

فایل کد: step4-3-5.py

```
#!/usr/bin/python3

import pprint
import itertools
from pymongo import MongoClient, DESCENDING

client = MongoClient()
db = client['bigdata99']

db.users.drop_indexes()

db.users.create_index([("location.state", DESCENDING)])

for index in db.users.list_indexes():
    print(index)
```

مجموعه ایندکس پیشنهادی:

میزان بهبود عملکرد: ۰,۶۴۷-۰,۶۵۷

توضیحات: با توجه به حجم کم داده‌ها، عملکرد index در این حالت چشمگیر نمی‌باشد.

بخش دوم

پیکربندی

در این بخش به نصب و پیکر بندی Neo4j پرداخته می‌شود. ابتدا می‌بایست به به نصب بسته openjdk-8-jre اقدام شود. نصب این نسخه لازمه اجرای Neo4j می‌باشد. پس از آن نصب نسخه Neo4j Community 3.5.18 به صورت tarball انجام می‌شود. این نسخه به صورت Standalone بوده و نیازی به نصب ندارد و تنها نیاز است تا دایرکتوری داخل فایل tar.gz. استخراج شده، و سپس دستور زیر در دایرکتوری استخراج شده در bash وارد شود:

```
$ ./bin/neo4j start
```

پس از این، دیتابیس Neo4j از طریق آدرس http://localhost:7474 در دسترس خواهد بود. در اولین ورود به دیتابیس لازم است تا کلمه عبور پیشفرض تغییر نماید. در اینجا کلمه عبور پیشفرض را برابر کاراکتر «a» وارد می‌کنیم.

در مرحله بعد، دیتاست Movies با یک میلیون رکورد دریافت شده است. در این مرحله با توجه به طولانی شدن فرآیند import دیتا به Neo4j توسط برنامه معرفی شده و همچنین عدم بهینه‌سازی‌های لازم در این برنامه، به صورت اختصاصی یک برنامه بهینه برای وارد کردن داده‌ها به دیتابیس نوشته است.

این برنامه از Driver رسمی پایتون Neo4j نسخه ۱,۷,۲ استفاده می‌کند. در ادامه سه Query برای وارد کردن دیتا به دیتابیس به صورت زیر نوشته شده است:

```
USING PERIODIC COMMIT 1000
LOAD CSV FROM "file:///movies.dat" AS csvLine FIELDTERMINATOR '\\t'
CREATE (m:Movie{{id: csvLine[0], title: substring(csvLine[1], 0,
    length(csvLine[1])-7), year: left(right(csvLine[1], 5), 4)}})
WITH csvLine, m
UNWIND split(csvLine[2], '|') AS genre
MERGE (g:Genre{{name: genre}})
CREATE (g)-[r:IS_GENRE_OF]->(m);
```

```
USING PERIODIC COMMIT 1000
LOAD CSV FROM "file:///users.dat" AS csvLine FIELDTERMINATOR '\\t'
CREATE (u:User{{id: csvLine[0], gender: csvLine[1], age:
    csvLine[2], occupation: csvLine[3], zipcode: csvLine[4]}});
```

```
USING PERIODIC COMMIT 1000
LOAD CSV FROM "file:///ratings.dat" AS csvLine FIELDTERMINATOR '\\t'
MATCH (u:User{{id: csvLine[0]}})
MATCH (m:Movie{{id: csvLine[1]}})
CREATE (u)-[r:RATED {{rating: csvLine[2], timestamp: csvLine[3]}}]->(m)
```

هر یک از این Query ها اقدام به وارد کردن دیتای یکی از سه فایل movies.dat، users.dat و ratings.dat به دیتابیس خواهد کرد. در این Query ها بهینه‌سازی‌های زیر رعایت شده است:

- استفاده از Periodic Commit برای کاهش حجم مصرفی حافظه و عدم انفجار داده‌ها در حافظه.
- استفاده از فرآیند LOAD CSV تعبیه شده در دیتابیس Neo4j که برای Batch Import بهینه شده است.
- تولید همزمان دو دسته Node از نوع Movie و Genre در یک Query با Parse نمودن جداگانه Genre های هر خط از فایل دیتاست مربوطه.
- جداسازی و Parse سال تولید فیلم و نام فیلم با استفاده از توابع خود Neo4j و بدون استفاده از برنامه جانبی برای پیش پردازش.

با استفاده از فرآیند import اختصاصی نوشته شده، کار وارد کردن داده‌ها به دیتابیس به حدود ۲ ساعت کاهش پیدا کرده است.

در این بین توجه به چند نکته لازم است:

- فایل‌های دیتاست لازم است تا درون دایرکتوری import در دایرکتوری اصلی دیتابیس Neo4j کپی شوند. آدرس دهی این فایل‌ها در Query به صورت نسبی از این دایرکتوری صورت خواهد گرفت.
- با توجه به عدم امکان Parse کردن فایل دیتا توسط Neo4j نسخه ۳٫۵ با استفاده از جداساز (Delimiter) چند حرفی، و با عنایت به اینکه فایل‌های دیتاست معرفی شده از جداساز «::» استفاده کرده‌اند (که یک جداسازی غیر معمول محسوب می‌گردد)، لازم است تا تمامی این جداسازها در فایل‌های دیتاست به یک کاراکتر جداساز یکتا تبدیل گردند. با توجه به عدم وجود کارکتر tab در فایل‌های دیتاست، پیش از اقدام به ورود دیتا به دیتابیس، با استفاده از دستور sed به تغییر کاراکتر جداساز فایل‌های دیتاست از «::» به tab اقدام می‌کنیم:

```
$ sed -i 's/::/\t/g' movies.dat
$ sed -i 's/::/\t/g' users.dat
$ sed -i 's/::/\t/g' ratings.dat
```

پس از این کار، و همچنین قرار دادن فایل‌های دیتاست در دایرکتوری ذکر شده، اقدام به اجرای فایل import.py خواهیم کرد:

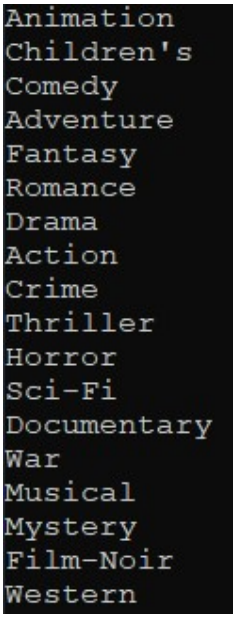
```
$ ./import.py
```

۱

پرس و جو:

```
MATCH (g:Genre)
RETURN g.name
```

خروجی:



توضیحات: در این بخش با دریافت تمامی نودهای از نوع ژانر (که در پرس و جوی مربوط به وارد کردن دیتا ایجاد شده است)، نام آن‌ها برگردانده می‌شود.

۲

پرس و جو:

```
MATCH (m:Movie)
RETURN count(m)
```

خروجی:

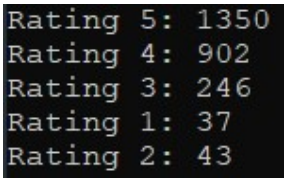
توضیحات: در این بخش تمامی نودها از نوع فیلم شمارش می‌گردند.

۳

پرس و جو:

```
MATCH ()-[r:RATED]->(m:Movie{{title: "Silence of the Lambs, The"}})
RETURN r.rating as rating, count(DISTINCT r) as count
```

خروجی:



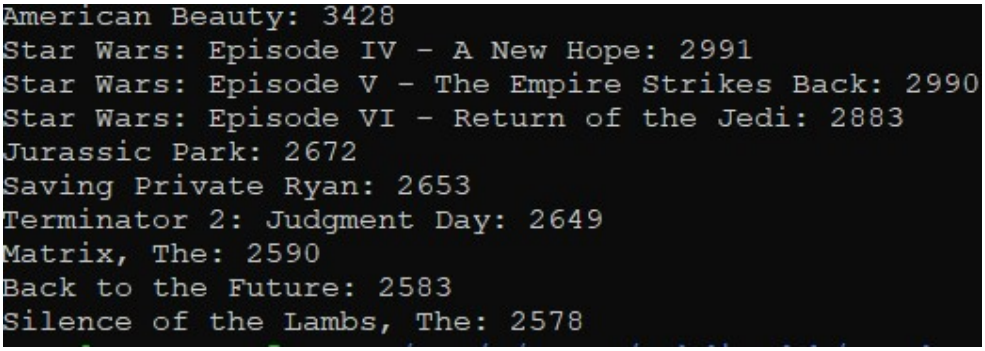
توضیحات: در این بخش با دریافت تمامی رابطه‌هایی از نوع RATED شامل نود مبدأ و مقصد، که نود مقصد نامی برابر فیلم سکوت بره‌ها دارد، امتیازات یکسان شمارش می‌شوند.

۴

پرس و جو:

```
MATCH ()-[r:RATED]->(m:Movie)
RETURN DISTINCT(m.title) as title, count(r) as count
ORDER BY count DESC
```

خروجی:



توضیحات: در این بخش با دریافت تمامی ارتباطات از نوع RATED به همراه نود مبدأ و مقصد، تعداد امتیازات هر فیلم شمارش شده و به صورت نزولی مرتب می‌گردند.

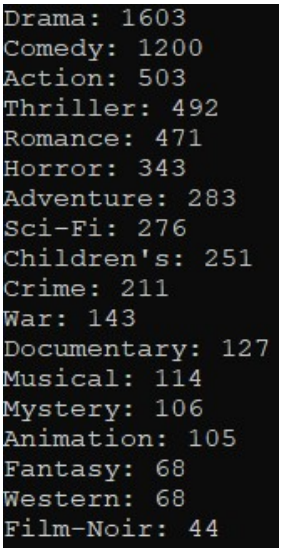
۵

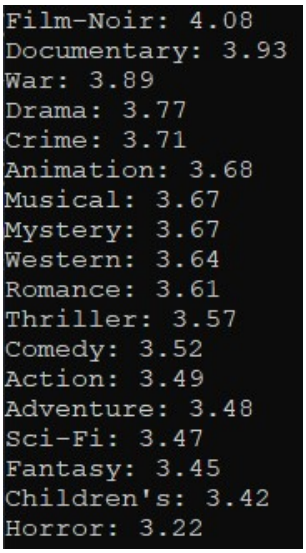
پرس و جو:

```
MATCH (g:Genre)-[i:IS_GENRE_OF]->(m:Movie)
RETURN DISTINCT(g.name) as genre, count(i) as count
ORDER BY count DESC

MATCH (g:Genre)-[i:IS_GENRE_OF]->(m:Movie)<-[r:RATED]-()
RETURN DISTINCT(g.name) as genre, round(avg(toFloat(r.rating))*100)/100 as average
ORDER BY average DESC
```

خروجی:

برای تعداد هر ژانر به صورت نزولی و همچنین میانگین امتیازات هر ژانر به صورت نزولی داریم:





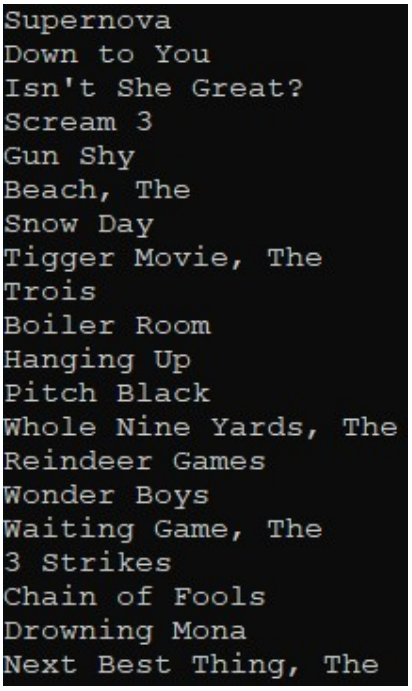
توضیحات: در این بخش با دریافت تمام ارتباطات ممکن بین ژانرها و فیلم‌ها، تعداد هر ژانر شمارش شده و به صورت نزولی مرتب می‌گردد. در قسمت دیگر، با دریافت تمامی ارتباطات مابین ژانر و فیلم و امتیاز دهندگان، لیست امتیازات هر ژانر بدست آمده که پس از آن از آن‌ها به ازای هر ژانر میانگین گرفته می‌شود.

۶

پرس و جو:

```
MATCH (m:Movie{{year: "2000"}})
RETURN m.title
```

خروجی:



در این بخش تمامی فیلم‌ها با سال تولید ۲۰۰۰ دریافت شده اند. فیلد سال در پرس و جوهای مربوط به وارد کردن داده به دیتابیس در قسمت‌های قبلی از نام فیلم جداسازی شده است.

۷

پرس و جو:

```
MATCH (u:User{{occupation: "12"}})-[r:RATED{{rating: "5"}}]->(m:Movie)
RETURN DISTINCT(m.title) as movie, count(r) as count
ORDER BY count DESC
```

خروجی:

```
Star Wars: Episode IV - A New Hope: 155
Star Wars: Episode V - The Empire Strikes Back: 128
Matrix, The: 120
Raiders of the Lost Ark: 117
American Beauty: 114
Sixth Sense, The: 104
Princess Bride, The: 96
Saving Private Ryan: 93
Blade Runner: 93
Fargo: 89
```

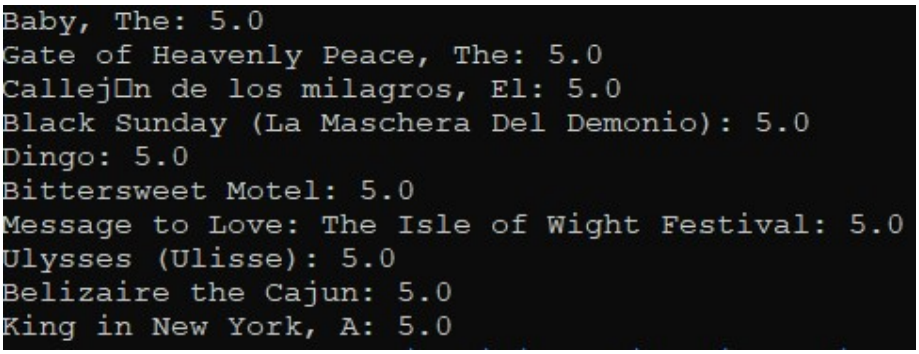
توضیحات: در این قسمت با دریافت تمامی کاربرانی که برنامه نویس هستند (کد ۱۲) و دریافت تمامی Ratingهای برابر ۵ ای که داده اند، به ازای هر فیلم شمارش شده و نتایج به صورت نزولی مرتب شده اند.

۸

پرس و جو:

```
MATCH (u:User)-[r:RATED]->(m:Movie)
WHERE u.age in ["18","25"]
RETURN DISTINCT(m.title) as movie, avg(toFloat(r.rating)) as average
ORDER BY average DESC
```

خروجی:



توضیحات: در این بخش با دریافت تمامی ارتباطات مابین کاربران رده سنی ۱۸ و ۲۵ سال که به فیلم‌ها امتیاز داده اند و فیلم ها، به ازای هر فیلم میانگین امتیازات داده شده بدست آمده است.

۹

پرس و جو:

```
MATCH (u:User)-[r:RATED]->(m:Movie)
WITH DISTINCT(m) as movie, count(r) as count
WHERE count >= 100
MATCH (uu:User)-[rr:RATED]->(movie)
RETURN DISTINCT(movie.title) as title, avg(toFloat(rr.rating)) as average
ORDER BY average DESC
```

خروجی:

توضیحات: در این بخش ابتدا تمامی روابط کاربران و فیلم‌ها استخراج شده، شرط تمامی فیلم‌هایی که تعداد امتیازات بیشتر از ۱۰۰ دارند به مرحله بعدی که میانگین امتیازات این فیلم‌ها را محاسبه و مرتب می‌کند منتقل می‌شود.

بخش سوم

پیکربندی

برای پیکربندی Hbase با توجه به عدم نیاز به انجام فاز production، آن را در Standalone Mode اجرا می‌کنیم.

برای اینکار فایل tar.gz. آخرین نسخه پایدار از Hbase را دریافت کرده (نسخه ۲,۲,۴) و آن را استخراج می‌کنیم. سپس در دایرکتوری حاصل و در فایل hbase-env.sh مقدار متغیر مربوط به محل نصب جاوا با نام JAVA_HOME را تنظیم می‌کنیم. در مرحله بعد، در فایل hbase-site.xml اقدام به درج موارد زیر می‌کنیم:

```
<configuration>
  <property>
    <name>hbase.rootdir</name>
    <value>file:///home/user/hbase</value>
  </property>
  <property>
    <name>hbase.zookeeper.property.dataDir</name>
    <value>/home/user/zookeeper</value>
  </property>
</configuration>
```

پس از این با اجرای دستور bin/start-hbase.sh دیتابیس راه اندازی خواهد شد. اینترفیس وب در آدرس localhost:16010 در دسترس بوده و با اجرای دستور bin/hbase shell امکان دسترسی به خط فرمان دیتابیس ممکن خواهد بود.

در این قسمت از کتابخانه happybase برای ارتباط با Hbase استفاده می‌کنیم. برای اینکار لازم است تا ابتدا سرور thrift راه اندازی شود. اینکار می‌تواند توسط دستور hbase thrift start انجام گردد.

گام اول

کد مربوط به این بخش در فایل import.py موجود می‌باشد. در این کد با ایجاد جدول‌های گفته شده به شرح زیر، اقدام به وارد کردن چند داده به صورت دستی در این جدول‌ها شده است.

در این مرحله ۳ جدول به شرح زیر طراحی می‌گردد. بدیهی است که طراحی هر جدول با هدف پاسخ دادن به پرسش‌های مطرح شده در گام بعدی طراحی شده است.

```
#!/usr/bin/python3

import happybase
connection = happybase.Connection('localhost')

connection.create_table(
    'movies',
    {
       'genres': {},
       'Mratings': {},
       'Fratings': {},
    }
)

table = connection.table('movies')

#Key: year_title
table.put(b'1995_Toy Story', {b'genres:Animation': b'', b"genres:Children's":
    b'', b'genres:Comedy': b'', b'Mratings:1':b'5', b'Mratings:2':b'4',
    b'Fratings:3':b'5', b'Fratings:4':b'3'})
table.put(b'1995_Jumanji', {b'genres:Adventure': b'', b"genres:Children's":
    '', b'genres:Fantasy': b'', b'Mratings:5':b'5', b'Mratings:6':b'2',
    b'Fratings:7':b'1', b'Fratings:8':b'4'})
table.put(b'1999_In Dreams', {b'genres:Thriller': b'', b'Mratings:1':b'3',
    b'Mratings:5':b'2', b'Fratings:7':b'1', b'Mratings:2':b'1'})

connection.create_table(
    'genres',
    {
       'movies': {}
    }
)

table = connection.table('genres')

table.put(b'Animation', {b'movies:Toy Story': b'1995'})
table.put(b"Children's", {b'movies:Toy Story': b'1995', b'movies:Jumanji':
    b'1995'})
table.put(b'Comedy', {b'movies:Toy Story': b'1995'})
table.put(b'Adventure', {b'movies:Jumanji': b'1995'})
table.put(b'Fantasy', {b'movies:Jumanji': b'1995'})
table.put(b'Thriller', {b'movies:In Dreams': b'1999'})


connection.create_table(
    'ratings',
    {
       'movies': {},
       'genres': {}
    }
)
```

```
table = connection.table('ratings')

#Key: age_user-id
table.put(b'01_1', {b'movies:Toy Story': b'5', b'movies:In Dreams': b'3'})
table.counter_inc(b'01_1', b'genres:Animation')
table.counter_inc(b'01_1', b"genres:Children's")
table.counter_inc(b'01_1', b'genres:Comedy')
table.counter_inc(b'01_1', b'genres:Thriller')

table.put(b'56_2', {b'movies:Toy Story': b'4', b'movies:In Dreams': b'1'})
table.counter_inc(b'56_2', b'genres:Animation')
table.counter_inc(b'56_2', b"genres:Children's")
table.counter_inc(b'56_2', b'genres:Comedy')
table.counter_inc(b'56_2', b'genres:Thriller')

table.put(b'25_3', {b'movies:Toy Story': b'5'})
table.counter_inc(b'25_3', b'genres:Animation')
table.counter_inc(b'25_3', b"genres:Children's")
table.counter_inc(b'25_3', b'genres:Comedy')

table.put(b'45_4', {b'movies:Toy Story': b'3'})
table.counter_inc(b'45_4', b'genres:Animation')
table.counter_inc(b'45_4', b"genres:Children's")
table.counter_inc(b'45_4', b'genres:Comedy')

table.put(b'25_5', {b'movies:Jumanji': b'5', b'movies:In Dreams': b'2'})
table.counter_inc(b'25_5', b'genres:Adventure')
table.counter_inc(b'25_5', b"genres:Children's")
table.counter_inc(b'25_5', b'genres:Fantasy')
table.counter_inc(b'25_5', b'genres:Thriller')

table.put(b'50_6', {b'movies:Jumanji': b'2'})
table.counter_inc(b'50_6', b'genres:Adventure')
table.counter_inc(b'50_6', b"genres:Children's")
table.counter_inc(b'50_6', b'genres:Fantasy')

table.put(b'35_7', {b'movies:Jumanji': b'1', b'movies:In Dreams': b'1'})
table.counter_inc(b'35_7', b'genres:Adventure')
table.counter_inc(b'35_7', b"genres:Children's")
table.counter_inc(b'35_7', b'genres:Fantasy')
table.counter_inc(b'35_7', b'genres:Thriller')

table.put(b'50_8', {b'movies:Jumanji': b'4'})
table.counter_inc(b'50_8', b'genres:Adventure')
table.counter_inc(b'50_8', b"genres:Children's")
table.counter_inc(b'50_8', b'genres:Fantasy')
```

فایل **import.py**

جدول movies

این جدول برای پاسخ دهی به سؤالات ۲، ۳ و ۵ گام دوم طراحی شده است. شماتیک این جدول به صورت زیر می‌باشد:

movies	genres:genre-name	Mratings:user-id	Fratings:user-id
year_title	empty	rating	rating

در این جدول کلیدها به فرم year_title قرار دارند (سال فیلم و نام فیلم). این امر با وجود قابلیت prefix scanning در Hbase امکان جستجوی سریع فیلم‌ها بر اساس سال تولید را فراهم می‌کند (قسمت ۵ گام دوم).

در این جدول ۳ خانواده ستون داریم. خانواده ستون اول با نام genres شامل اطلاعات ژانر فیلم خواهد بود. مقدار qualifier در این خانواده ستون برابر ژانر فیلم خواهد بود. این امر پاسخگویی به قسمت ۲ گام دوم را تسهیل می‌سازد.

خانواده ستون بعدی با نام Mratings بوده که شامل امتیازات داده شده توسط کاربران مرد به این فیلم می‌باشد. مقدار qualifier در این خانواده برابر شناسه کاربر و همچنین مقدار سلول برابر امتیاز داده شده به فیلم خواهد بود. خانواده ستون Fratings نیز به همین صورت شامل امتیازات داده شده کاربران زن به این فیلم می‌باشد. این طراحی بابت تسهیل در پاسخگویی به قسمت ۵ گام دوم انجام شده است. همچنین در قسمت ۳ گام دوم نیز دریافت همزمان این دو خانواده ستون ممکن بوده و پاسخ آن را تأمین خواهد کرد.

جدول genres

این جدول برای پاسخ دهی به سؤال ۱ گام دوم طراحی شده است. شماتیک این جدول به صورت زیر می‌باشد:

genres	movies:movie-title
name	year

در این جدول کلیدها به صورت نام ژانرها می‌باشند. خانواده ستون movies شامل qualifierهای با نام فیلم بوده و مقدار سلول‌ها شامل سال تولید فیلم خواهند بود.

جدول ratings

این جدول برای پاسخگویی به سؤال ۴ گام دوم طراحی شده است. شماتیک این جدول به صورت زیر می‌باشد:

ratings	movies:movie-title	genres:genre-name
age_id	rating	counter

کلیدها در این جدول به فرم سن-شناسه می‌باشند. این امر با وجود قابلیت prefix scanning در Hbase امکان جستجوی سریع کاربران بر اساس رده سنی آن‌ها را فراهم می‌کند (قسمت ۴ گام دوم).

در این جدول دو خانواده ستون داریم. خانواده ستون اول شامل نام فیلم‌ها به صورت qualifier بوده که مقدار هر سلول امتیاز داده شده توسط کاربر به آن می‌باشد. خانواده ستون دوم شامل ژانرها می‌باشد. در این خانواده ستون هر ژانر به صورت qualifier بوده و مقدار سلول برابر تعداد دفعات امتیاز داده شده توسط کاربر به فیلمی از آن ژانر می‌باشد که با هر بار امتیاز دهی، به صورت عملیات atomic، یک عدد به شمارنده سلول مربوط به آن ژانر اضافه خواهد شد. این امر پاسخگویی به بخش ۴ گام دوم را تسهیل می‌سازد.

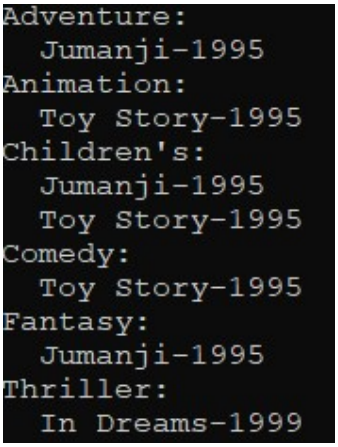
گام دوم

۱

کد: (فایل 1.py)

```
#!/usr/bin/python3
import happybase
connection = happybase.Connection('localhost')
table = connection.table('genres')
for key, data in table.scan(columns=[b'movies']):
    print(f'{key.decode()}:')
    for record in data:
       print(f'  {record.decode().split(":")[1]}-{data[record].decode()}')
```

خروجی:



توضیحات: در این بخش با دریافت کل خانواده ستون movies از جدول genres، برای هر ژانر فیلم‌های مربوطه بدست آورده شده اند.

۲

کد: (فایل 2.py)

```
#!/usr/bin/python3
import happybase
connection = happybase.Connection('localhost')
table = connection.table('movies')
for key, data in table.scan(columns=[b'genres']):
    print(f'{key.decode()}: {len(data)}')
```

خروجی:

```
1995_Jumanji: 3
1995_Toy Story: 3
1999_In Dreams: 1
```

توضیحات: در این بخش با دریافت خانواده ستون genres از جدول movies، تعداد ژانرها برای هر فیلم با شمارش آن‌ها انجام می‌شود. دقت شود که در اینجا استفاده از counter نیز ممکن بود، ولی با توجه به امکان درخواست پرس و جوی دریافت نام ژانرهای هر فیلم، شمارش ژانرها به تنهایی مناسب به نظر نمی‌رسد.

۳

کد: (فایل 3.py)

```
#!/usr/bin/python3
import happybase
connection = happybase.Connection('localhost')
table = connection.table('movies')
ratings = {}
for key, data in table.scan(columns=[b'Mratings',b'Fratings']):
    ratings[key.decode()] = len(data)
for key, value in sorted(ratings.items(), key=lambda item: item[1], reverse
    = True):
    print(f'{key}: {value}')
```

خروجی:

```
1995_Jumanji: 4
1995_Toy Story: 4
1999_In Dreams: 4
```

توضیحات:

در این بخش با دریافت خانواده ستون‌های امتیازدهندگان مرد و زدن به صورت همزمان و در کنار هم، اقدام به شمارش و مرتب سازی این امتیازات شد. در اینجا نیز امکان استفاده از counter محیا بود، ولی با توجه به نیاز به خود امتیازات در مراحل بعدی، از استفاده از counter صرف نظر به عمل آمد.

۴

کد: (فایل 4.py)

```
#!/usr/bin/python3
import happybase
import collections
connection = happybase.Connection('localhost')
table = connection.table('ratings')
ratings = {}
ages = [b'45',b'50',b'56']
movies = []
genres = {}
for age in ages:
     for key, data in table.scan(row_prefix=age,columns=[b'movies']):
        for record in data:
               movies.append(record.decode().split(":")[1])
counter=collections.Counter(movies)
print("Popular Movies:")
for key, value in sorted(dict(counter).items(), key=lambda item: item[1], reverse =
     True):
     print(f' {key}: {value}')
for age in ages:
     for key, data in table.scan(row_prefix=age,columns=[b'genres']):
        for record in data:
               if record.decode().split(":")[1] not in genres:
                      genres[record.decode().split(":")[1]] =
     int.from_bytes(data[record], "big")
               else:
                      genres[record.decode().split(":")[1]] =
     genres[record.decode().split(":")[1]] + int.from_bytes(data[record], "big")
counter=collections.Counter(genres)
print("Popular Genres:")
for key, value in sorted(dict(counter).items(), key=lambda item: item[1], reverse =
     True):
     print(f' {key}: {value}')
```

خروجی:

```
Popular Movies:
 Toy Story: 2
 Jumanji: 2
 In Dreams: 1
Popular Genres:
 Children's: 4
 Animation: 2
 Comedy: 2
 Adventure: 2
 Fantasy: 2
 Thriller: 1
```

توضیحات: در این بخش ابتدا با استفاده از row prefix، بازه‌های سنی هدف از جدول دریافت شده و تمامی امتیازات فیلم‌ها کنار هم گردآوری شده شده است و محبوب‌ترین آن‌ها اعلام گشته است. همچنین برای دریافت ژانرهای محبوب از counter استفاده شده که با دریافت عدد هر یک، گردآوری و مرتب سازی آن‌ها این مقدار نیز اعلام شده است.

۵

کد: (فایل 5.py)

```
#!/usr/bin/python3
import happybase
connection = happybase.Connection('localhost')
table = connection.table('movies')
Mratings = []
Fratings = []
for key, data in table.scan(row_prefix=b'1995',columns=[b'Mratings']):
    for record in data:
       Mratings.append(int(data[record].decode()))
for key, data in table.scan(row_prefix=b'1995',columns=[b'Fratings']):
    for record in data:
       Fratings.append(int(data[record].decode()))
print(f'Average rating of 1995 movies by males:
    {sum(Mratings)/len(Mratings)}')
print(f'Average rating of 1995 movies by females:
    {sum(Fratings)/len(Fratings)}')
```

خروجی:

```
Average rating of 1995 movies by males: 4.0
Average rating of 1995 movies by females: 3.25
```

توضیحات:

در این بخش با دریافت جداگانه خانواده ستون‌های شامل امتیازات مرد و زن به فیلم، و همچنین با استفاده از امکان row prefix برای دریافت فیلم‌های یک سال مشخص، اقدام به میانگین گیری برای هر یک از این دسته‌ها صورت گرفته است.