به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

کلان داده

تمرین شماره 2

?

اردیبهشت ماه 1399

فهرست گزارش سوالات

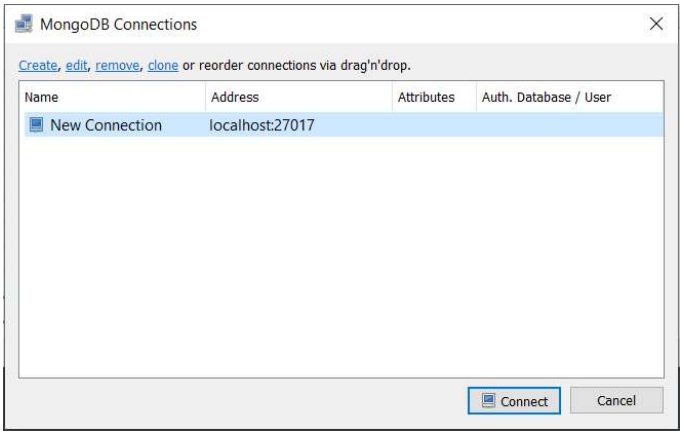
بخش اول – ذخیره اطلاعات بدون ساختار/کار با MongoDB

در این بخش از تمرین می‌خواهیم با پایگاه داده‌های MongoDB آشنا شویم. MongoDB از جمله پایگاه داده‌های مبتنی بر Document می‌باشد. ساختار کلی این پایگاه داده به این صورت است که مجموعه-ای از Documentها تشکیل Collection را می‌دهد و مجموعه‌ای از Collectionها در نهایت پایگاه داده ما را تشکیل خواهند داد. در بخش‌های مختلف این سوال به تفصیل درباره این پایگاه داده و نحوه استفاده از آن صحبت خواهیم کرد.

گام اول تمرین – ایجاد دیتابیس

در این قسمت ابتدا نسخه مربوط به ویندوز نرم‌افزار MongoDB را از سایت رسمی آن دانلود می‌نماییم و نصب می‌کنیم. فرآیند نصب بسیار ساده بوده و از لینک داده شده در صورت سوال جهت راهنمایی استفاده شده‌است. جهت سهولت در کار با پایگاه داده MongoDB ما در اجرای این تمرین از نرم‌افزارهای گرافیکی آن استفاده نموده‌ایم. نرم‌افزار Robo 3T به علت سادگی برای این کار انتخاب شده‌است.

در ابتدا با فراخوانی آدرس https://randomuser.me/api/?nat=ir به تولید تصادفی اطلاعات کاربران می‌پردازیم. با هر بار بارگذاری مجدد این صفحه یک کاربر جدید به صورت تصادفی ایجاد خواهد شد. در گام اول می‌خواهیم یک دیتابیس نمونه جهت ذخیره سازی تعدادی از این اطلاعات ایجاد نماییم. برای این کار از نرم‌افزار Robo 3T استفاده می‌کنیم. در هنگام ورود به این برنامه نیاز به ایجاد یک Connection جهت اتصال به دیتابیس داریم. برای این منظور با انتخاب گزینه Connect در منوی File با صفحه زیر مواجه خواهیم شد.

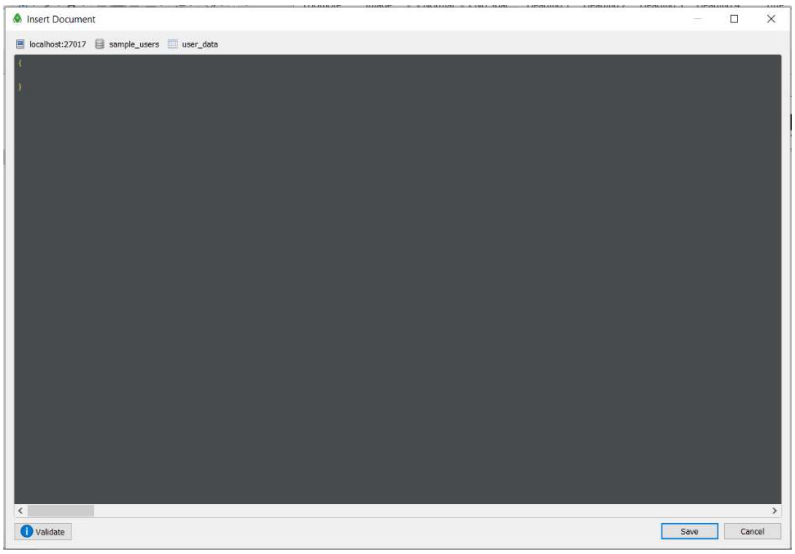


شکل 1 : صفحه مربوط به ساخت اتصال به دیتابیس در برنامه Robo 3T

در صفحه مورد نظر با انتخاب گزینه create در بالای آن می‌توانیم یک Connection جدید ایجاد نماییم که در شکل 1، یک نمونه از آن را مشاهده می‌کنید. برای ایجاد این اتصال نیاز به یک Address وجود دارد که نام آن را localhost و پورت آن را 27017 انتخاب می‌کنیم. این پورت معمولا در پایگاه داده MongoDB مورد استفاده قرار می‌گیرد و در نهایت با فشردن کلید connect به سرور مورد نظر متصل خواهیم شد.

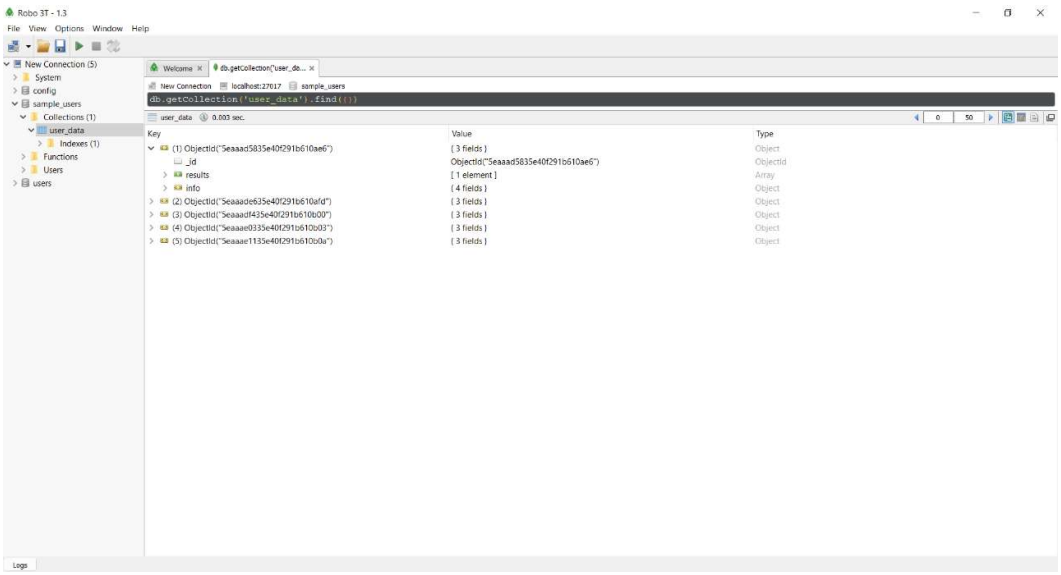
در گام اول یک دیتابیس نمونه ایجاد می‌نماییم. برای ایجاد دیتابیس روش‌های مختلفی وجود دارد که ما در اینجا به این صورت عمل می‌کنیم که در ابتدا بر روی New Connection که در منوی سمت راست نرم‌افزار وجود دارد راست کلیک کرده و گزینه create database را انتخاب می‌نماییم. در این مرحله نیاز به انتخاب یک نام برای آن داریم که ما در اینجا آن را sample_users می‌نامیم. بعد از این مرحله در زیر شاخه sample_users پوشه‌ای به نام Collections وجود دارد که اطلاعات ما باید در این بخش ذخیره شود، برای این منظور بر روی این پوشه کلیک راست کرده و گزینه create collection را انتخاب می‌کنیم و با انتخاب نامی دلخواه که در اینجا user_data می‌باشد، Collection مورد نیاز را ایجاد می‌کنیم. حال با انتخاب user_data در Collection، می‌توانیم محتویات آن را که شامل Documentهای ما می‌باشد، مشاهده کنیم. البته در ابتدای کار این بخش شامل داده‌ای نمی‌باشد.

در ادامه می‌خواهیم به صورت دستی به تعداد 5 نمونه اطلاعات کاربران تولید شده در سایت Randomuser را به این پایگاه داده اضافه نماییم. برای این منظور با انجام راست کلیک در صفحه مربوط به user_data با گزینه‌ای به نام Insert Document مواجه می‌شویم که با فشردن آن صفحه‌ای مانند شکل2 باز خواهد شد.



شکل2: پنجره مربوط به ورود اطلاعات کاربران در قالب Json

در گام بعد اطلاعاتی که توسط سایت Randomuser تولید شده است را در این پنجره کپی می‌نماییم و در نهایت گزینه Save را انتخاب می‌کنیم. این کار را می‌توانیم به دفعات انجام دهیم. ما در اینجا برای 5 کاربر این کار را انجام داده‌ایم. البته می‌توان اطلاعات را به کمک فرامین مربوط به این پایگاه داده نیز اضافه نمود که ما در اینجا راه ساده‌تر را انتخاب نموده‌ایم. در نهایت با اضافه کردن اطلاعات این پنج کار، ساختار User-data به صورت زیر خواهد شد.



شکل3: اضافه شدن اطلاعات 5 کاربر در دیتابیس Sample_Users

نکته‌ای که وجود دارد این است که دیتابیس MongoDB به اطلاعات هر کاربر یک فیلد جدید به نام ID اضافه می‌نماید. این فیلد برای هر داده کاملا منحصر به فرد می‌باشد که از آن جهت کار با شاخص‌ها یا Indexing استفاده می‌کند. در مورد این مفهوم به تفصیل در گام چهارم تمرین توضیح خواهیم داد. بقیه بخش‌های داده بدون هیچ تغییری و با همان ساختار تعریف شده، در پایگاه داده MongoDB ذخیره می‌شوند.

حال که با مقدمات کار آشنا شدیم، می‌خواهیم در ادامه به ساخت پایگاه داده اصلی که شامل اطلاعات صدهزار کاربر می‌باشد بپردازیم. برای این منظور کد پایتونی ایجاد نموده‌ایم که این کد در پوشه کدها و در زیرپوشه Part1 و با نام Get_Database_Save.py می‌باشد. پیش از توضیح کد باید به این نکته توجه نماییم که برای تولید داده‌ها تصادفی از سایت Randomuser محدودیتی وجود دارد که در هر گام فقط می‌توان 5000 کاربر تولید نمود و به صورت متوالی فقط اطلاعات 20000 کاربر را در اختیار قرار می‌دهد. طبق گفته خود سایت در واقع برای جلوگیری از شلوغی پهنای باند و استفاده دیگران این کار صورت گرفته

است و متاسفانه حداقل نیاز به 3 دقیقه زمان وجود دارد تا بتوان 20000 داده بعدی را از سایت درخواست نمود. برای همین منظور ما در کد پایتون یک حلقه با اندازه 20 بار تکرار ایجاد نموده‌ایم که در هر بار 5000 داده را می‌خواند و بعد از هر 4 مرتبه خواندن به مدت 3 دقیقه منتظر می‌مانیم تا مجددا بتوانیم درخواست خود را برای 20000 داده بعدی اعمال نماییم. برای ایجاد این تاخیر از تابع countdown() استفاده نموده‌ایم.

برای خواندن 5000 اطلاعات کاربری از سایت مورد نظر از قطعه کد زیر استفاده نموده‌ایم.

```
url = "https://randomuser.me/api/"
querystring = {"nat":"ir", "results":"5000"}
response = requests.request("GET", url, params=querystring)
data = json.loads(response.text)
```

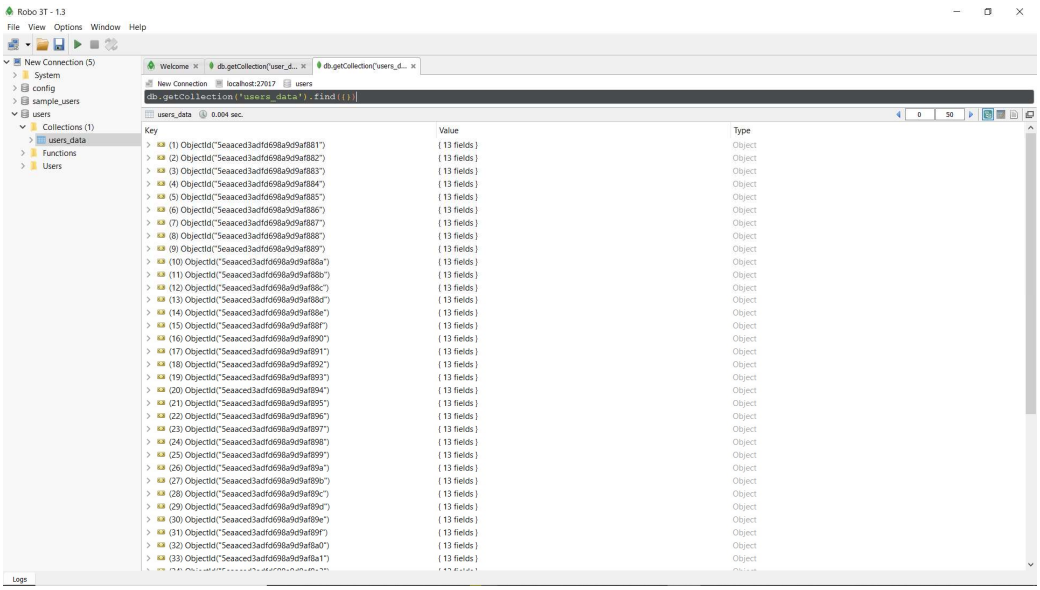
خروجی این قسمت data می‌باشد که به صورت یک dictionary و دارای دو بخش است، بخش مربوط به results، لیستی از اطلاعات مربوط به کاربران را در خود دارد و ما از همین بخش به صورت data['results'] برای ذخیره در پایگاه داده استفاده می‌کنیم. قسمت دیگر آن Info می‌باشد که مربوط به تعداد داده‌ها خوانده شده و غیره می‌باشد.

در بخش بعد به کمک کتابخانه PyMongo ابتدا به Client مورد نظر که قبلا ایجاد نمودیم متصل می‌شویم و سپس دیتابیسی به نام users می‌سازیم و در نهایت به کمک تابعی به نام insert_many() اطلاعات را به Collection مربوط به User_data اضافه می‌نماییم. کد مربوط به این توضیحات به صورت زیر است.

```
client = MongoClient('localhost', 27017)
db = client['users']
users_data = db.users_data
users_data.insert_many(data['results'])
```

در انتهای کار به کمک دستور زیر تعداد Documentهایی که در پایگاه دادهٔ ایجاد شده وجود دارد را می‌شماریم که با اجرای آن مقدار 100000 را نمایش می‌دهد. البته برای بررسی صحت این بخش از شیوه دیگری نیز استفاده می‌کنیم که در ادامه درباره آن توضیح خواهیم داد.

حال که دیتابیس را ایجاد نمودیم مجددا به برنامه Robo 3T باز می‌گردیم تا از ساخت این دیتابیس اطمینان پیدا نماییم. با ورود به این برنامه با نام دیتابیس users در منوی سمت راست مواجه خواهیم شد که با ورود به پوشه Collections در آن محتویات User_data را خواهیم دید.



شکل4: شمایی از دیتابیس ساخته شده با نام Users

همانطور که در شکل بالا مشاهده می‌نمایید، نوار مشکی رنگی وجود دارد که در واقع یک Shell می‌باشد که می‌توانیم به راحتی تمامی دستورات مربوط به پایگاه داده MongoDB را در آن اجرا نماییم. اولین دستوری که می‌خواهیم در این بخش در آن اجرا نماییم مربوط به شمارش سندها جهت صحت وجود صد هزار سند می‌باشد. برای این منظور از کد زیر استفاده می‌نماییم.

```
db.users_data.count({})
```

با اجرای این کد مقدار صد هزار به عنوان خروجی نمایش داده می‌شود که نشان از بارگزاری درست تمامی داده‌ها در دیتابیس مورد نظر می‌باشد. در این دستور db به پایگاه داده مورد نظر اشاره دارد و در بخش بعد user_data مربوط به Collectionی می‌باشد که می‌خواهیم عملیات بر روی آن انجام شود و در نهایت تابع Count() بدون هیچ آرگومانی صدا زده می‌شود که خالی بودن آن بدین معنی است که هیچ شرطی بر روی نحوه شمارش داده‌ها اعمال نشده‌است. که خروجی اجرای آن همانطور که پیش‌تر گفتیم برابر 100000 می‌باشد.

گام دوم تمرین – دستورات اصلی

در این قسمت از تمرین به بررسی دستورات پایه در مانگو دی‌بی می‌پردازیم. تمامی کدهای نوشته شده در این قسمت به زبان پایتون و به کمک کتابخانه Pymongo صورت گرفته‌است که در ادامه بیشتر با آن‌ها آشنا خواهیم شد.

سوال 1)

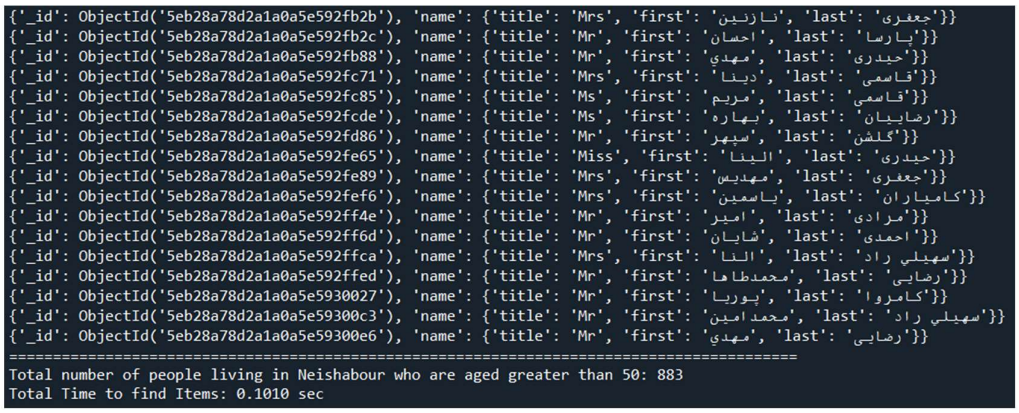
در این قسمت می‌خواهیم نام و نام‌خانوادگی افرادی که ساکن نیشابور و سن آن‌ها بالاتر از 50 سال می‌باشد را بدست آوریم. برای این منظور بعد از اینکه به دیتابیس مورد نظر متصل شدیم، به کمک دستور find() و اعمال شرط مورد نظر بر روی آن، درخواست داده مورد نظر را صادر می‌نماییم. قسمت اصلی کد مربوط به آن به صورت زیر می‌باشد.

```
out_data = []
for results in user_data.find({"location.city": "نیشابور",
                               "dob.age": {'$gt' : 50}},
                              {'name':1}):
        out_data.append(results)
        print(results)
```

همانطور که در کد بالا مشاهده می‌نمایید، تابع Find()، دو بخش مختلف دارد که در قسمت اول شرط مربوط به جستجو اعمال شده است که می‌خواهیم شهر مربوط به افراد نیشابور باشد و سن آن‌ها از 50 سال بیشتر باشد که به کمک $gt این شرط برقرار خواهد شد و در قسمت بعد داده‌هایی که باید برگردانده شوند مشخص می‌شود که در اینجا ما فقط اطلاعات مربوط به نام افراد با این مشخصات را می‌خواهیم. حلقه For که در اینجا نوشته شده است به این دلیل است که معمولا درخواست Find() به دیتابیس MongoDB یک رشته از داده‌ها را باز می‌گرداند که برای دسترسی و نمایش تک تک این داده‌های حاصل از این درخواست نیاز به اعمال این حلقه می‌باشیم و ارتباطی به درخواست مورد نظر ندارد و صرفا برای نمایش داده‌ها مورد استفاده قرار می‌گیرد.

حال با اجرای کد مربوط به این بخش که در پوشه HW2_Step1 و به نام Part2_Q1.py می‌باشد. نام تمامی افراد در خروجی نمایش داده می‌شود که نمونه‌ای از آن را در زیر مشاهده می‌کنید.

خروجی برنامه



مدت زمان اجرای این درخواست به صورت کلی برابر 101 میلی‌ثانیه می‌باشد که در مجموع 883 نفر با این مشخصات را در دیتابیس مورد نظر، پیدا نموده‌است.

سوال 2)

در این قسمت به دنبال اطلاعات پایه از افرادی هستیم که بیش از بیست سال است در سایت ما ثبت نام کرده‌اند. این اطلاعات نیز به صورت ساده به کمک یک درخواست به شکل Find بدست خواهد آمد که کد مربوط به آن را در زیر مشاهده می‌نمایید.

```
for results in user_data.find({"registered.age" : {'$gt' : 17}},
                              {"name.last":1,
                               "location.street":1,
                               "location.city":1,
                               "location.state":1,
                               "location.country":1,
                               "location.postcode":1,
                               "cell":1,
                               "registered.age":1}):
        out_data.append(results)
        print(results)
```

همانطور که در کد بالا مشاهده می‌نمایید، قسمت سبز رنگ مربوط به درخواست ما می‌باشد که قسمت اول آن مربوط به شرط سوال و افراد با سابقه بیست سال می‌باشد. نکته‌ی قابل ذکر این است که متاسفانه در دیتابیس مورد نظر فردی با سابقه بیش‌تر از 18 سال وجود نداشت و برای اینکه درخواست مورد نظر

ما خروجی بازگرداند مجبور شدم که به جای سابقه بیست سال از سابقه 18 سال استفاده نمایم که در درخواست فوق در بخش اول تابع Find شرط بزرگتر از 17 سال را بروی سابقه ثبت نام اعمال نموده‌ایم و در قسمت دوم اطلاعاتی که باید از دیتابیس بازگردانده شود را مشخص نموده‌ایم. این اطلاعات به ترتیب شامل نام‌خانوادگی، جزئیات اطلاعات پستی و شماره موبایل می‌باشد. در زیر بخش از این خروجی را مشاهده می‌نمایید.

خروجی برنامه

```
{'_id': ObjectId('5eb28a78d2a1a0a5e59301d8'), 'name': {'last': 'محمدخان'}, 'location': {'street':
27983}, 'registered': {'age': 18}, 'cell': '0901-747-8238'}
{'_id': ObjectId('5eb28a78d2a1a0a5e59301e6'), 'name': {'last': 'قاسمی'}, 'location': {'street': {'
40429}, 'registered': {'age': 18}, 'cell': '0974-084-1024'}
{'_id': ObjectId('5eb28a78d2a1a0a5e59301fb'), 'name': {'last': 'نکو نظر'}, 'location': {'street':
'Iran', 'postcode': 64512}, 'registered': {'age': 18}, 'cell': '0982-800-5313'}
{'_id': ObjectId('5eb28a78d2a1a0a5e59301fc'), 'name': {'last': 'علیزاده'}, 'location': {'street':
21728}, 'registered': {'age': 18}, 'cell': '0945-434-8190'}
{'_id': ObjectId('5eb28a78d2a1a0a5e593020d'), 'name': {'last': 'کامیاران'}, 'location': {'street':
46786}, 'registered': {'age': 18}, 'cell': '0905-793-3677'}
{'_id': ObjectId('5eb28a78d2a1a0a5e5930224'), 'name': {'last': 'نجاتی'}, 'location': {'street': {'
'postcode': 68685}, 'registered': {'age': 18}}
================================================================================
Total number of people who have the register time greater than 17: 4513
Total Time to find Items: 0.3969 sec
```

همانطور که در شکل بالا مشاهده می‌نمایید، تعداد 4513 نفر دارای سابقه 18 سال ثبت نام را در سایت ما دارند و این درخواست به زمانی برابر 396.9 میلی‌ثانیه برای پاسخ نیاز داشته‌است. کد مربوط به این بخش در پوشه کدها و در زیرپوشه HW2_Step1 و با نام Part2_Q2.py موجود می‌باشد.

سوال 3)

در این قسمت می‌خواهیم در دو بخش از دیتابیس، سال میلادی را به سال شمسی تبدیل نماییم و در نهایت یک آیتم جدید به هر کدام از این بخش‌ها به نام year_persian اضافه نماییم. برای انجام این بخش روش‌ها مختلفی وجود دارد که ما در اینجا به ترتیب زیر عمل کرده‌ایم.

در ابتدا به کمک دستور ()Find و بدون اعمال هیچ شرطی تاریخ تولد و تاریخ ثبت نام افراد را می‌خوانیم و در متغیرهایی به نام dob_date و reg_date ذخیره می‌نماییم. در گام بعد به کمک تابعی به نام ()convert_miladi_to_shamsi این دو تاریخ را به تاریخ شمسی تبدیل می‌نماییم. تابع مورد نظر، سال، ماه و روز را دریافت می‌نماید و به کمک یک الگوریتم استاندارد در این زمینه، سال، ماه و روز شمسی را باز می‌گرداند. حال به کمک دستور ()Update به اضافه نمودن آیتم جدید به دیتابیس می‌پردازیم. دستور Update به این صورت است که برای هر کاربر یا Id، به کمک دستور $set فیلد جدیدی به نام

Year_persian را در زیرمجموعه dob و registered اضافه می‌نماییم و مقادیر بدست آمده در قسمت قبل را به آن‌ها نسبت می‌دهیم. کد مربوط به این بخش در پوشه کدها و در زیرپوشه HW2_Step1 و به نام Part2_Q3.py می‌باشد که بخش اصلی این کد که مربوط به درخواست می‌باشد را در زیر مشاهده می‌نمایید.

```
for results in user_data.find({}, {'dob.date':1, 'registered.date':1}):
  dob_date = results['dob']['date']
  reg_date = results['registered']['date']
  dob_year, dob_month, dob_day =
       convert_miladi_to_shamsi(int(dob_date[0:4]), int(dob_date[5:7]), int(dob_date[8:10]))
  reg_year, reg_month, reg_day =
      convert_miladi_to_shamsi(int(reg_date[0:4]), int(reg_date[5:7]), int(reg_date[8:10]))
  user_data.update_one({'_id': results['_id']},
                       {'$set': {"dob.year_persian":str(dob_year)+'-'+str(dob_month)+'-
                                 '+str(dob_day)+dob_date[10:],
                                "registered.year_persian": str(reg_year)+'-
                                 '+str(reg_month)+'-'+str(reg_day)+reg_date[10:]}})
```

این کد به علت اینکه نیاز به محاسبات مربوط به تبدیل سال میلادی به شمسی را برای هر کاربر باید انجام دهد، نیاز به زمان بیشتری دارد و برای بروزرسانی تمامی کاربران حدود 36.14 ثانیه زمان نیاز دارد. در ادامه برابر بررسی صحت تغییرات انجام‌شده به کمک نرم‌افزار Robo 3T به بررسی آیتم اضافه شده می‌پردازیم که خروجی آن را در تصویر زیر مشاهده می‌نمایید.

شکل5: نمونه‌ای از آیتم‌های اضافه شده به دیتابس، حاصل از اجرای کد موردنظر

سوال 4)

در این قسمت می‌خواهیم در پایان هر روز به افرادی که روز تولدشان می‌باشد، یک ایمیل تبریک ارسال نماییم، برای این منظور نیاز به نام و نام‌خانوادگی و ایمیل این افراد داریم. جهت انجام این کار می‌توانیم از دستور ساده Find() استفاده نماییم. کد مربوط به این بخش در پوشه کدها و در زیرپوشه HW2_Step1 و به نام Part2_Q4.py می‌باشد. بخش اصلی این کد که مربوط به درخواست می‌باشد را در زیر مشاهده می‌نمایید.

```
pattern = '.*-'+m+'-'+d+'.*'

out_data = []
for results in user_data.find({'dob.date': {'$regex': pattern}},
                              {"name.first":1,
                               "name.last":1,
                               "email":1,
                               "dob.date":1}):
    out_data.append(results)
    print(results)
```

همانطور که در کد بالا مشاهده می‌کنید، از دستور Find() استفاده شده است که بر روی تاریخ تولد یک شرط اعمال شده‌است. تاریخ تولد که با متغیر dob.date نمایش داده شده است به صورت String در دیتابیس ذخیره شده‌است برای اینکه تاریخ تولد فرد را با تاریخ امروز مقایسه نماییم از دستور $regix استفاده نموده‌ایم که امکان کار با String را برای ما فراهم می‌سازد. برای این منظور در ابتدا تاریخ روز را به کمک توابع آماده date می‌خوانیم و سپس ماه و روز را به یک pattern به صورت ‘.*-month-day.*’ تبدیل می‌نماییم که مثلا برای یک تاریخ مشخص به صورت ".*-05-15.*" می‌باشد که معنی آن این است که در رشته مربوط به تاریخ تولد بررسی می‌نماید و هر رشته‌ای که شامل این ساختار باشد را باز می‌گرداند و به این ترتیب می‌توانیم افرادی را که تاریخ تولدشان در این روز می‌باشد را پیدا نماییم و در قسمت دوم فقط اطلاعات مربوط به نام و نام‌خانوادگی، ایمیل و در نهایت تاریخ تولد را جهت اعتبارسنجی بازمی‌گردانیم. با اجرای کد مورد نظر خروجی مانند آنچه که در ادامه بیان شده‌است می‌باشد که برای اجرا و پردازش این درخواست به زمانی در حدود 168 میلی‌ثانیه نیاز می‌باشد. همچنین تعداد افرادی که تاریخ تولد آن‌ها در این روز می‌باشد هم به عنوان خروجی نمایش داده شده‌است.

خروجی برنامه

```
{'_id': ObjectId('5eb28a78d2a1a0a5e592f7b9'), 'name': {'first': 'آیلین', 'last': 'کوتی'}, 'email': 'aylyn.khwty@example.com',
{'_id': ObjectId('5eb28a78d2a1a0a5e592fa31'), 'name': {'first': 'سارینا', 'last': 'زارعی'}, 'email': 'sryn.zraay@example.com',
{'_id': ObjectId('5eb28a78d2a1a0a5e592fcd2'), 'name': {'first': 'آراد', 'last': 'گلشن'}, 'email': 'ard.glshn@example.com', 'do
{'_id': ObjectId('5eb28a78d2a1a0a5e592fd0a'), 'name': {'first': 'محمدپارسا', 'last': 'سلطانی نژاد'}, 'email': 'mhmdprs.sltnyn
{'_id': ObjectId('5eb28a78d2a1a0a5e592fdd1'), 'name': {'first': 'مهدی', 'last': 'سلطانی نژاد'}, 'email': 'mhdy.sltnynjd@examp
{'_id': ObjectId('5eb28a78d2a1a0a5e592fe8e'), 'name': {'first': 'مارال', 'last': 'صدر'}, 'email': 'mrl.sdr@example.com', 'dob'
{'_id': ObjectId('5eb28a78d2a1a0a5e59301d8'), 'name': {'first': 'آدرین', 'last': 'محمدخان'}, 'email': 'adryn.mhmdkhn@example.c
================================================================================
Total number of people that today is their birthday: 282
Total Time to find Items: 0.1688 sec
```

سوال 5)

در این قسمت می‌خواهیم درباره نحوه ذخیره ایمن Password صحبت نماییم. همانطور که در دیتاست مشاهده می‌نمایید، در قسمت Login عبارت مربوط به Password قابل مشاهده می‌باشد و این موضع معمولا از لحاظ امنیت بسیار نامناسب می‌باشد. معمولا در هنگام ذخیره گذرواژه از شیوه‌های Hashing استفاده می‌شود و به جای اینکه مستقیما خود گذرواژه در دیتابیس ذخیره شود، گذروازه از یک تابع Hash عبور می‌نماید و سپس معادل آن یک عبارت به شکل‌های باینری، هگزادسیمال و غیره تولید می‌شود که از لحاظ ظاهری هیچ شباهتی به گذرواژه اصلی ندارد. در این شرایط زمانی که کاربر گذرواژه خود را وارد می‌نماید این عملیات مجددا بر روی آن انجام می‌گیرد و با کد تولید شده مقایسه می‌شود و در صورت برابری، اجازه ورود کاربر داده می‌شود.

با توجه به نکات بیان شده، معمولا شیوه‌ها یا کدینگ‌های مختلفی برای این منظور وجود دارد که معمولا با نام‌هایی مثل Md5، SHA1، SHA256 و غیره می‌باشد.

حال برای این بخش ما می‌توانیم شیوه‌های مختلفی را مورد استفاده قرار دهیم. شیوه اول این است که بدون در نظر گرفتن اطلاعات موجود در دیتابیس و به کمک کتابخانه bcrypt در پایتون به خواندن گذرواژه‌ها از دیتابیس بپردازیم و سپس به کمک توابع آماده‌ای مانند getsalt() عمل Encryption را بر روی گذرواژه انجام دهیم و خروجی آن را که یک رشته باینری طولانی می‌باشد، به جای گذرواژه ذخیره نماییم ولی این فرآیند می‌تواند اندکی زمانگیر باشد. خوشبختانه با توجه به اطلاعات موجود در دیتابیس مانند md5، salt و غیره، می‌توان از همین اطلاعات بابت عمل کدینگ گذرواژه استفاده نماییم. در ادامه شیوه این کار توضیح خواهیم داد.

کد مربوط به این بخش در پوشه کدها و در زیرپوشه HW2_Step1 و به نام Part2_Q5.py موجود می‌باشد. در ادامه بخش مربوط به درخواست اولیه جهت تغییر گذرواژه‌ها را مشاهده می‌نمایید.

```
user_data.update_many({},[{'$set': {"login.password":"$login.salt"}}])
```

با اجرای درخواست بالا بر روی دیتابس مورد نظر بعد از حدود 10.6383 ثانیه، تمامی گذرواژه‌های مربوط به کاربران با عبارت موجود در بخش login.salt جایگزین می‌شود. با این کار امکان نمایش گذرواژه‌ها به صورت کلی از بین خواهد رفت و به جای آن یک رشته کاراکتر بی معنی جایگزین می‌شود. حال در ادامه دلیل این امر را توضیح خواهیم داد.

در دیتابیس مورد نظر در قسمت login داده‌ای به نام md5 وجود دارد که ما از آن برای عمل Hashing گذرواژه استفاده می‌نماییم. برای این منظور از کتابخانه آماده hashlib در پایتون استفاده می‌نماییم. عملکرد آن به این صورت است که گذرواژه را توسط کاربر از ورودی دریافت می‌نماید و آن را به صورت utf-8 تبدیل می‌نماید و سپس رشته مربوط به salt را که در دیتابیس وجود دارد به آن اضافه می‌نماید و در نهایت به کمک تابع hashlib.md5().hexdigest() به رشته md5 تبدیل می‌نماید که در صورت درست بود گذرواژه مقدار تولید شده با مقدار موجود در دیتابیس برابر می‌باشد. به همین علت ما در دیتابیس به جای گذرواژه‌ها مقدار Salt را ذخیره نموده‌ایم. کد مربوط به این بخش را در زیر مشاهده می‌نمایید.

```
user = input('Please Enter Your Username :')
pasw = input('Please Enter Your Password :')
results = user_data.find_one({'login.username': user},
                             {'name':1,
                              'login.username':1,
                              'login.password':1,
                              'login.md5':1})
if results != None:
   pass_string = pasw.encode() + results['login']['password'].encode()
   hash_out = hashlib.md5(pass_string).hexdigest()
   if hash_out == results['login']['md5']:
       print(">>>> Password match <<<<")
       print('Real Name :', results['name'])
       print('User Name :', results['login']['username'])
   else:
       print("Error: Password didn't match !!!")
else:
   print("Error: User Name didn't Match !!!")
```

کد بالا عملیات چک کردن نام کاربری و گذرواژه می‌باشد. در ابتدا از کاربر می‌خواهد که نام کاربری و گذرواژه خود را وارد نماید و سپس به کمک درخواست Find() به دنبال نام کاربری مورد نظر می‌رود و در صورت وجود، اطلاعاتی شامل نام فرد، نام کاربری، گذرواژه و مقدار md5 آن را باز می‌گرداند. سپس مقدار

گذرواژه خوانده شده از دیتابیس را که همان Salt می‌باشد به گذرواژه ورودی اضافه کرده و به تابع تولید md5 می‌دهد. در نهایت این رشته تولیدی با رشته موجود در دیتابیس مقایسه می‌شود و در صورت برابر اطلاعات کاربر چاپ شده و در غیر این صورت آن را اعلام می‌نماید.

به عنوان مثال اطلاعات کاربری فردی در این دیتابیس به این صورت است که نام کاربری آن "happygoose207" و گذواژه آن برابر "clifford" می‌باشد. با ورود این اطلاعات خروجی برنامه به صورت زیر می‌باشد.

خروجی برنامه

```
Please Enter Your Username :happygoose207

Please Enter Your Password :clifford

>>>> Password match <<<<
Real Name : {'title': 'Mr', 'first': 'آرش', 'last': 'رضاییان'}
User Name : happygoose207
```

حال اگر مثلا نام گذواژه را به صورت غلط وارد نماید، خروجی برنامه به صورت زیر می‌باشد که اعلام اشتباه بودن گذرواژه را می‌نماید.

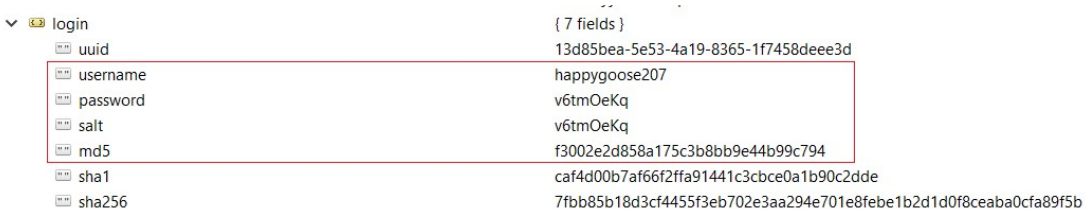
خروجی برنامه

```
Please Enter Your Username :happygoose207

Please Enter Your Password :test

Error: Password didn't match !!!
```

در پایان نیز در شکل زیر نمونه‌ای از مقادیر جایگذاری شده با گذرواژه در دیتابیس را مشاهده می‌نمایید.



شکل6: وضعیت گذرواژه در دیتابیس بعد از اجرای درخواست Update_many()

گام سوم تمرین – دستورات تجمعی و آماری

تا به اینجای کار با دستورات پایه مانگو آشنا شدیم، در این قسمت می‌خواهیم با دستورات تجمعی آشنا شویم. این دستورات امکانات بسیار زیادی را جهت کار با پایگاه‌داده MongoDB فراهم می‌سازند که در ادامه با بخشی از این ویژگی‌ها آشنا خواهیم شد.

سوال 1)

در این قسمت می‌خواهیم به کمک دستورات تجمعی، کاربران را به سه دسته زیر 16 سال، بین 16 تا 30 سال و بالای 30 سال تقسیم نماییم و تعداد هر دسته را به عنوان خروجی این بخش بازگردانیم.

کد مربوط به این قسمت در پوشه کدها و در زیرپوشه HW2_Step1 و به نام Part3_Q1.py می‌باشد. برای این منظور از درخواست زیر استفاده می‌کنیم که در ادامه بیشتر درباره آن توضیح خواهیم داد.

```
results = user_data.aggregate([{
        "$facet": {
            'Group1':[
                {'$match': {'dob.age': {'$lte': 16}}},
                {'$count': "counter"}],
            'Group2':[
                {'$match': {'$and': [{'dob.age': {'$gt': 16}},
                                     {'dob.age': {'$lte': 30}}]}},
                {'$count': "counter"}],
            'Group3':[
                {'$match': {'dob.age': {'$gt': 30}}},
                {'$count': "counter"}]
        }},{
        "$project": {
            "Group1": { "$arrayElemAt": ["$Group1.counter", 0] },
            "Group2": { "$arrayElemAt": ["$Group2.counter", 0] },
            "Group3": { "$arrayElemAt": ["$Group3.counter", 0] }}}
    ]).next()
```

همانطور که در بالا مشاهده می‌کنید، از دستور Aggregate() استفاده نموده‌ایم که در ابتدا از دستور $facet برای اجرای همزمان و موازی چندین درخواست استفاده می‌کنیم. در ادامه سه درگروه درخواست Group1، Group2 و Group3 را داریم که در هر کدام ابتدا به کمک دستور $match به نوعی شرط گروه مورد نظر را تعیین می‌کنیم و به کمک $count تعداد دفعاتی که آن شرط مهیا می‌شود را می‌شماریم. در گروه اول شمارش در حالتی که dob.age کمتر و مساوی 16 باشد صورت می‌گیرد، در گروه دوم حالتی که بین 16 تا 30 باشد را می‌شماریم و در نهایت در گروه سوم مقدار سن بالای 30 را می‌شماریم. در پایان به کمک $Project نیز نحوه خروجی را مشخصی می‌نماییم که این قسمت فقط جهت شکل دهی بهتر به

خروجی می‌باشد که نبود آن نیز، خللی در انجام درخواست مورد نظر ایجاد نخواهد کرد. با اجرای کد بالا، خروجی به صورت زیر می‌باشد.

خروجی برنامه

```
{'Group2': 16248, 'Group3': 83752}
============================================================
Total Time to find Items: 0.7375 sec
============================================================
Total number of people who are under 16         :  0
Total number of people who are between 16 to 30 :  16248
Total number of people who are above 30         :  83752
Total number of people in the database          :  100000
```

همانطور که مشاهده می‌نمایید، اجرای درخواست فوق بر روی دیتاست مورد نظر به مدت 737 میلی-ثانیه به طول می‌انجامد.

سوال 2)

در این قسمت می‌خواهیم تعداد کاربران هر استان را به تفکیک بدست آوریم. کد مربوط به این قسمت در پوشه کدها و در زیرپوشه HW2_Step1 و به نام Part3_Q2.py قابل مشاهده می‌باشد. قسمت اصلی این کد را که مربوط به درخواست مورد نظر می‌باشد در زیر مشاهده می‌نمایید.

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

همانطور که در کد بالا مشاهده می‌نمایید، از خاصیت $group استفاده نموده‌ایم. در واقع متغیر location.state اشاره به استان‌ها دارد و به کمک دستور $group که بر روی این متغیر اعمال می‌شود، باعث می‌شود که کاربران هر استان را در یک گروه قرار دهد و در نهایت به کمک خط دوم که دستور $sum در آن وجود دارد، تعداد اعضای هر گروه شمارش شود و با اجرای آن متغیر Results نتیجه را در قالب عددی به عنوان تعداد اعضای هر استان باز می‌گرداند که به تعداد استان‌ها در آن عضو وجود دارد. در ادامهٔ کد برای نمایش بهتر نتایج اقدام به چاپ آن در قالب بهتر و به کمک دستورات پایتون نموده‌ایم که

ارتباطی به درخواست مورد نظر ندارد و صرفا برای نمایش اطلاعات در قالب بهتر می‌باشد. در زیر قسمتی از خروجی حاصل از اجرای این کد را مشاهده می‌نمایید.

خروجی برنامه

```
State20        3201                البرز
State21        3240                بوشهر
State22        3174                مرکزی
State23        3198                خوزستان
State24        3239                کرمان
State25        3126          آذربایجان شرقی
State26        3147                اصفهان
State27        3188                یزد
State28        3211                قزوین
State29        3166                گلستان
State30        3230                مازندران
State31        3326       چهارمحال و بختیاری

Total People :  100000
============================================
Total Time to find Items: 0.3830 sec
```

همانطور که در بالا مشاهده می‌نمایید، برای هر استان، تعداد کاربران در جلوی آن چاپ شده‌است که در بالا فقط بخشی از استان‌ها نمایش داده شده‌اند. زمان اجرای این درخواست برابر 383 میلی‌ثانیه می‌باشد.

سوال 3)

در این قسمت می‌خواهیم برای هر فردی که بخش آفست مربوط به timezone آن برابر 5:00+ می‌باشد، فیلد مربوط به cell را در آن حذف نماییم و سپس تعداد افرادی که شماره موبایل ندارند را پیدا نماییم. کد مربوط به این قسمت در قسمت کدها و در زیرپوشه HW2_Step1 و به نام Part3_Q3.py قابل مشاهده می‌باشد. قسمت اصلی کد مربوط به این بخش را در زیر مشاهده می‌نمایید.

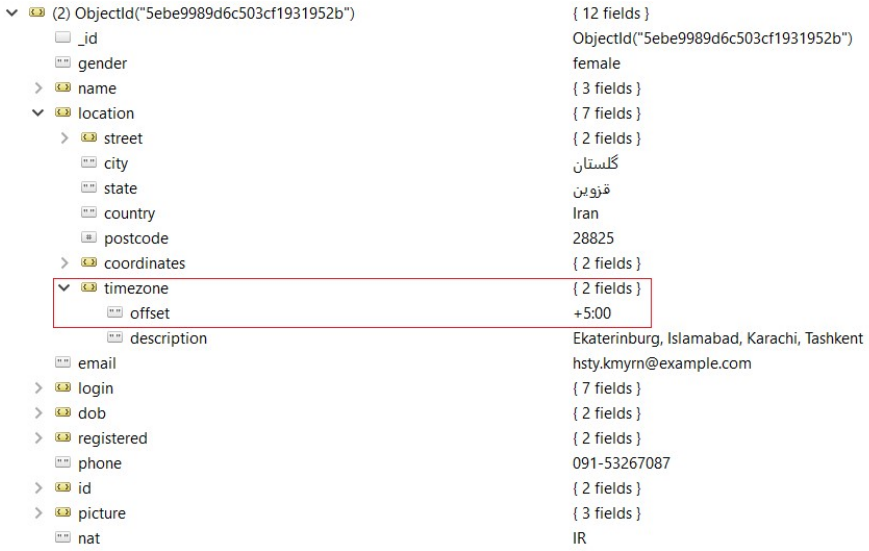
```
user_data.update_many({'location.timezone.offset' : "+5:00"},
                      {'$unset': {'cell':""}})

total = list(user_data.aggregate([
                      {'$match': {'cell': {'$exists': 'false'}}},
                      {'$count': 'no_cell_cnt'}]))
```

همانطور که در کد بالا مشاهده می‌نمایید، در قسمت اول به کمک دستور Update_many() به حذف فیلد مربوط به موبایل برای کاربران مورد نظر پرداخته‌ایم. در قسمت اول آن با اعمال شرط بر روی متغیر location.timezone.offset و مقایسه آن با مقدار 5:00+ و در صورت برابری به کمک دستور $unset فیلد

مربوط به cell را که به موبایل اشاره دارد به صورت کلی حذف می‌نماییم. حال در درخواست بعدی تعداد افرادی را که شماره موبایل یا فیلد مربوط به cell را ندارند می‌شماریم. برای این کار به این صورت عمل می‌کنیم که از دستورات تجمعی استفاده می‌کنیم و به کمک $exists بر روی فیلد مربوط به cell شرط اعمال می‌نماییم و به این ترتیب حتی مواردی که در آن‌ها فیلد Cell وجود دارد ولی مقداری در آن نیست نیز در اینجا در نظر گرفته می‌شود و به این ترتیب عمل شمارش را انجام خواهیم داد.

با اعمال درخواست اول جهت حذف کردن فیلد cell از دیتابس مورد نظر، وضعیت جدید دیتابیس برای فیلدهای مورد نظر به صورت زیر خواهد شد.



شکل7: وضعیت فیلد cell در دیتابس مورد نظر بعد از اجرای درخواست بالا

همانطور که در شکل بالا مشاهده می‌نمایید، برای کاربر مورد نظر که آفست زمانی مورد نظر سوال را دارد، فیلدی به نام Cell وجود ندارد و حذف شده‌است.

با اجرای کد مورد نظر خروجی برنامه به صورت زیر می‌باشد که در آن تعداد فیلدهایی که شماره موبایل در آن‌ها وجود ندارد را نمایش داده‌است. همچنین مدت زمان اجرای این درخواست برابر 413 میلی‌ثانیه می‌باشد.

خروجی برنامه

```
Total number of people who do not have cellphone:  3286
================================================================
Total Time to execute query: 0.4130 sec
```

سوال 4)

در این قسمت می‌خواهیم میانگین سنی کاربران تهرانی را نسبت به میانگین سنی کاربران دیگر استان‌ها مقایسه نماییم. کد مربوط به این قسمت در پوشه کدها و در زیر پوشه HW2_Step1 و به نام Part3_Q4.py قابل مشاهده می‌باشد. بخش مربوط به درخواست‌های مورد نظر در این کد را در زیر مشاهده می‌نمایید.

```
tehran_state = user_data.aggregate([
                {'$match': {"location.state": "تهران"}},
                {'$group': { '_id': "Tehran",
                             'age_sum': {'$sum': '$dob.age'},
                             'user_cnt': {'$sum': 1},
                             'average': {'$avg': '$dob.age'}}}]).next()

other_state = user_data.aggregate([
               {'$match': {"location.state": {'$ne': "تهران"}}},
               {'$group': { '_id': "Others",
                            'age_sum': {'$sum': '$dob.age'},
                            'user_cnt': {'$sum': 1},
                            'average': {'$avg': '$dob.age'}}}]).next()
```

در درخواست اول، ابتدا میانگین سنی کاربران تهرانی را بدست آورده‌ایم. در ابتدا به کمک دستور $match بروی متغیر location.state شرطی بر اساس استان تهران اعمال نموده‌ایم و در قسمت بعد به محاسبه میانگین پرداخته‌ایم. در قسمت دوم که از $group استفاده نموده‌ایم، می‌توانستیم ساختار را ساده‌تر بنویسیم ولی ما در اینجا برای اینکه از صحت عملکرد متوسط‌گیری مطمئن شویم، مقادیر مربوط به جمع کل سنین و تعداد کل کاربران را نیز محاسبه نموده‌ایم که با مشاهده و تقسیم این دو بر هم از صحت مقدار متوسط مطمئن شویم و در واقع نیازی به این کار نبوده‌است و در پایان مقدار متوسط را به کمک $avg و اعمال آن بر روی متغیر dob.age بدست آورده‌ایم. در درخواست دوم نیز روال کار به همین صورت است، با این تفاوت که در دستور $match به کمک $ne نقیض شرط برابر استان تهران را اعمال نموده‌ایم که به این ترتیب بقیه استان‌ها در نظر گرفته شده‌اند. خروجی حاصل از اجرای این کد را در زیر مشاهده می‌نمایید.

خروجی برنامه

```
Average Age of Users in Tehran State: 48.4181
Average Age of Users in Other States: 48.7798

Total Users :  100000
============================================================
Total Time to find Items: 0.6599 sec
```

همانطور که در شکل بالا مشاهده می‌نمایید، زمان اجرای درخواست‌ها فوق برابر 659 میلی‌ثانیه می‌باشد.

سوال 5)

در این قسمت می‌خواهیم بررسی نماییم که کدام شهر بیشترین کاربر و کدام شهر کمترین کاربر را دارد. کد مربوط به این قسمت در پوشه کدها و در زیرپوشه HW2_Step1 و به نام Part3_Q5.py می‌باشد. در زیر، بخش اصلی این کد را که مربوط به ارسال درخواست مورد نظر می‌باشد، مشاهده می‌نمایید.

```
results = list(user_data.aggregate([
          {'$group': {'_id': "$location.city", 'count': {'$sum': 1}}},
          {'$sort': {'count': 1}}]))
```

همانطور که در بالا مشاهده می‌نمایید، به کمک دستور $group و بر اساس متغیر location.city، اقدام به گروه‌بندی افراد بر اساس شهر محل زندگی آن‌ها خواهیم پرداخت و در هر گروه به کمک $sum و مانند ساختار فوق به شمارش افراد هر شهر می‌پردازیم. سپس در گام بعدی به کمک دستور $sort مقادیر شمارش شده را به صورت صعودی که با 1 نمایش داده شده‌است مرتب می‌نماییم و به این ترتیب با مشاهده results که خروجی مورد نظر می‌باشد، عضو اول شهر با کمترین کاربر و آخرین عضو نیز شهر با بیشترین

خروجی برنامه

```
The Most  Users are in ساری and total number of them are 2115
The Least Users are in شیراز and total number of them are 1865
================================================================================
Total Time to find Items: 0.3138 sec
```

کاربر را نشان می‌دهد. خروجی مربوط به این کد را در ادامه مشاهده می‌نمایید.

همانطور که در بالا مشاهده می‌نمایید، زمان اجرای این درخواست در حدود 313 میلی‌ثانیه می‌باشد.

گام چهارم تمرین – بررسی کارایی شاخص‌ها

در این قسمت می‌خواهیم به کمک اضافه نمودن Index به دیتابیس مورد نظر، روند جستجو در دیتابیس را بهینه نماییم. تا به اینجای کار تنها ایندکس _id در دیتابیس وجود داشت و برای هر درخواستی عملیات جستجو و حرکت در دیتابیس بر اساس همین پارامتر و بر روی کل داده‌ها صورت می‌گیرفت ولی خوشبختانه امکان اضافه نمودن ایندکس‌های مختلف به دیتابیس وجود دارد که به کمک آن‌ها می‌توانیم زمان درخواست‌ها را بهبود دهیم. البته باید به این نکته نیز اشاره نماییم که افزایش تعداد ایندکس‌ها خود باعث پیچیدگی می‌شود و معمولا دیتابیس‌ها خود یک محدودیتی در تعداد آن‌ها اعمال می‌نماید. در ادامه برخی از سوالات موجود در بخش‌های قبل را به کمک ایندکس‌گذاری مجددا اجرا خواهیم کرد و اثر آن‌ها را بر روی زمان اجرای درخواست‌ها مشاهده می‌کنیم. تمامی کدهای مورد نظر در پوشه کدها و در زیرپوشه HW2_Step1 و با پیشوند Part4 قابل مشاهده می‌باشند.

گام دوم-سوال 1)

در این قسمت می‌خواهیم درخواست افراد ساکن نیشابور و با سن بیشتر از 50 سال را مجددا با ایندکس گذاری مناسب اجرا نماییم. در حالت معمول تنها ایندکسی که وجود داشت به نام _id بود که اطلاعات خاصی را در خود نداشت و بنابراین درخواست مورد نظر مجبور بود که تمامی Documentها را بررسی نماید و فیلدهای سن و شهر آن‌ها را چک نماید تا بتواند به درخواست مورد نظر پاسخ دهد. در قسمت‌های قبل دیدیم که این درخواست در حدود 101 میلی‌ثانیه به طول می‌انجامد. حال می‌خواهیم یک ایندکس ایجاد نماییم که این فرآیند را بهبود دهد. در محیط پایتون و به کمک Pymongo این کار را می‌توان به وسیله تابع create_index() انجام داد. البته به کمک برنامه Robo 3T و به صورت گرافیکی و مراجعه به قسمت Index ها در هر Collection می‌توان ایندکس مورد نظر خود را اضافه یا حذف نماییم. در زیر ایندکس مناسب برای این کار را مشاهده می‌کنید.

```
user_data.create_index([('dob.age', pymongo.DESCENDING),
                        ('location.city', pymongo.DESCENDING)],
                         name='index1')
```

برای این منظور از یک ایندکس ترکیبی استفاده می‌نماییم. پارامترهای این ایندکس سن و شهر می‌باشند که به صورت نزولی مرتب شده‌اند و دلیل آن این است که اولا ما به دنبال افراد با سن بالای 50 سال می‌گردیم، بنابراین اگر به صورت نزولی مرتب شود تمامی عناصر از مقدار ماکزیمم به مقدار مینیمم در

دسترس می‌باشند و ما کافیست فقط تا سن 50 سال را جستجو نماییم و همچنین برای بهبود بیشتر نام شهر را نیز به صورت نزولی اعمال می‌نماییم و دلیل آن این است که شهر نیشابور در همان موارد ابتدایی قرار می‌گیرد و به این ترتیب فضای جستجو به جای صد هزار کاربر بسیار محدودتر خواهد شد. با اجرای کد مورد نظر، زمان اجرا حدودا 40 میلی‌ثانیه خواهد شد که نشان از بهبود عملکرد می‌باشد.

گام دوم-سوال 4)

در این قسمت می‌خواهیم افرادی را که امروز رو تولدشان می‌باشد، جستجو نماییم. در حالت معمول دیدیم که زمان اجرای این درخواست برابر 168 میلی‌ثانیه می‌باشد. برای بهبود عملکرد این درخواست از کد زیر استفاده می‌نماییم.

```
user_data.create_index([('dob.date', pymongo.TEXT)], name='date_index')


user_data.find({'$text': {'$search': pattern}},
               {"name.first":1, "name.last":1, "email":1, "dob.date":1})
```

از آنجایی که ما در اینجا با تاریخ تولد سرو کار داریم و این تاریخ در متغیر dob.date قرار دارد و از نوع String می‌باشد، بنابراین برای بهبود عملکرد از Text Indexing استفاده می‌نماییم. برای این منظور در ابتدا به کمک Create_index() متغیر مورد نظر را از نوع text تعریف می‌نماییم و در گام بعد در تابع find() به راحتی به کمک $text و $search به دنبال الگوی مورد نظر در dob.date می‌پردازیم. در اینجا Pattern در واقع ماه و روز مود نظر می‌باشد که در تاریخ تولد باید جستجو شود. با اعمال موارد فوق مدت زمان درخواست دستور فوق برابر 78 میلی‌ثانیه می‌باشد.

گام دوم-سوال 5)

در این سوال به بهبود نحوه ذخیره گذرواژه می‌پردازیم. روشی که ما در اینجا اعمال نمودیم این بود که به جای مقدار موجود در فیلد Password مقدار مربوط به فیلد salt را قرار دادیم. برای این منظور چون باید بروی تمامی Documentهای موجود در Collection این عمل را انجام دهیم، بنابراین به نظر می‌رسد

که مفهوم ایندکس چندان کاربرد ندارد ولی برای قسمتی که با وارد کردن نام کاربری و گذرواژه، می‌خواهیم عملیات اعتبارسنجی را انجام دهیم، می‌توانیم از Indexing استفاده نماییم. برای این منظور چون عملیات بر روی نام کاربری صورت می‌گیرد، می‌توانیم نام کاربری را به صورت مثلا صعودی مرتب نماییم تا اینکه احتمالا بخشی از جستجو مورد ارزیابی قرار نگیرد.

```
user_data.create_index([('login.username', pymongo.ASCENDING)],
                       name='user_index')
```

گام سوم-سوال2)

در این قسمت می‌خواهیم تعداد کاربران هر استان را به تفکیک بدست آوریم. برای این منظور پیش‌تر ملاحظه فرمودید که از دستور $group و $sum برای این منظور استفاده نمودیم که انجام این درخواست به مدت 383 میلی‌ثانیه به طول می‌انجامد. از آنجایی که در این‌جا ما نیاز داریم که عمل شمارش بر روی تمام دیتابیس صورت گیرد، بنابراین اندکس‌گذاری شاید چندان مناسب این حالت نباشد مگر اینکه ما در دیتابیس Documentهای داشته باشیم که فیلد مربوط به شهر آن‌ها بدون مقدار یا اصلا وجود نداشته باشد که در این صورت در حالت معمول ما تمام آن‌ها را چک می‌نماییم. بنابراین برای چنین مواردی از Index به شکل زیر استفاده می‌نماییم.

```
user_data.create_index([('location.state', pymongo.ASCENDING)],
                       name='state_index', sparse = True)
```

بر روی Location.state عمل اندکس‌گذاری صورت گرفته است و در انتها با فعال نمودن گزینه Sparse این امکان را فراهم می‌نماییم که فقط اسنادی که فیلد مورد نظر را دارند مورد ارزیابی قرار گیرند. البته در اینجا چون تمامی اسناد مقدار Location.state را دارند، این ایندکس عملکرد خود را به خوبی نشان نمی‌دهد ولی در واقعیت معمولا اینگونه نمی‌باشد.

گام سوم-سوال5)

در این قسمت می‌خواهیم بررسی نماییم که کدام شهر بیشترین کاربر و کدام شهر کمترین کاربر را دارد. در قسمت‌های قبل به کمک $group و سپس $sort مقادیر مورد نظر را محاسبه نمودیم و دیدیم که درخواست مورد نظر به زمانی در حدود 313 میلی‌ثانیه نیاز دارد. حال می‌خواهیم ایندکس مناسب برای این کار را بدست آوریم.

گام پنجم تمرین – استفاده از ساختار Map/Reduce

در این قسمت می‌خواهیم به کمک ویژگی Map/Reduce به محاسبه میانگین سنی افراد به تفکیک هر شهر بپردازیم. کد مربوط به این قسمت در پوشه کدها و در زیرپوشه HW2_Step1 و با نام Part5.py قرار دارد. در زیر کد مربوط به فرآیند Map/Reduce را مشاهده می‌نمایید.

```
map = Code("function() {"
           "var key = this.location.city;"
           "var value = {count: 1, age: this.dob.age };"
           "emit(key, value);"
           "};")

reduce = Code("function(key, values) {"
              "reducedVal = { count: 0, age: 0 };"
              "for (var idx = 0; idx < values.length; idx++) {"
                   "reducedVal.count += values[idx].count;"
                   "reducedVal.age += values[idx].age;"
              "}"
              "return reducedVal;"
              "};")

final = Code("function (key, reducedVal) {"
             "reducedVal.avg = reducedVal.age/reducedVal.count;"
             "return reducedVal"
             "};")

result = user_data.map_reduce(map, reduce, "myresults", finalize=final)
```

در محیط پایتون و به کمک کتابخانه bson.code امکاناتی وجود دارد که به وسیله تابع Code() می‌توانیم تمامی دستورانی که برای دیتابیس MongoDB وجود دارد را عینا و در بین " " بنویسیم و اجرا نماییم.

در ابتدا تابعی به نام map ایجاد می‌نماییم که در این تابع مقدار کلید برابر شهر می‌باشد که متغیر location.city به آن اشاره دارد و Value در اینجا زوج سن و مقدار 1 می‌باشد. بنابراین فرآیندی که در پروسه Map صورت می‌گیرد این است که در هر Document به ازای شهر به عنوان کلید، مقدار سن کاربر و عدد یک را نسبت می‌دهد و آن را به بخش Reduce ارسال می‌نماید.

تابع Reduce به این صورت عمل می‌کند که کلید و مقدار را دریافت می‌نماید و در ابتدا متغیری به نام ReduceVal که به صورت زوج Count و Age می‌باشد، می‌سازد، سپس در یک حلقه به طول بردارValue به صورت مجزا جمع سنین و تعداد آن‌ها را بدست می‌آورد. نکته مهم اینجا این است که بر اساس خاصیت

Map/Reduce این فرآیند بر اساس هر شهر به تفکیک انجام می‌شود، چا که کلید ما در اینجا شهرها می‌باشند. حال خروجی Reduce به تابع Final که خود نوعی Reduce می‌باشد ارسال می‌شود و این تابع نیز کلید و مقدار مرحله قبل را دریافت کرده و با تقسیم تعداد کل افراد به جمع سنین، مقدار متوسط سن افراد بدست می‌آید و بازگردانده می‌شود و این تابع نیز با توجه به مقدار کلید این فرآیند را به تفکیک هر شهر انجام می‌دهد.

حال در گام آخر به کمک تابع آماده map_reduce() در دیتابیس MongoDB، توالی اجرای این توابع را مشخص می‌نماییم که در ابتدا تابع Map، بعد از آن تابع Reduce و در گام آخر تابع Final انجام می‌شود که نتایج نیز در قالب myResults می‌باشد. خروجی این تابع در متغیر results ریخته می‌شود که متوسط سن افراد به تفکیک هر شهر را نشان می‌دهد که در ادامه به کمک توابع موجود در پایتون به نمایش آن‌ها پرداخته‌ایم.

از آنجایی که خروجی برنامه از 50 شهر مختلف تشکیل شده‌است، ما در ادامه بخشی از خروجی این کد را که شامل فقط 20 شهر می‌باشد، نمایش می‌دهیم.

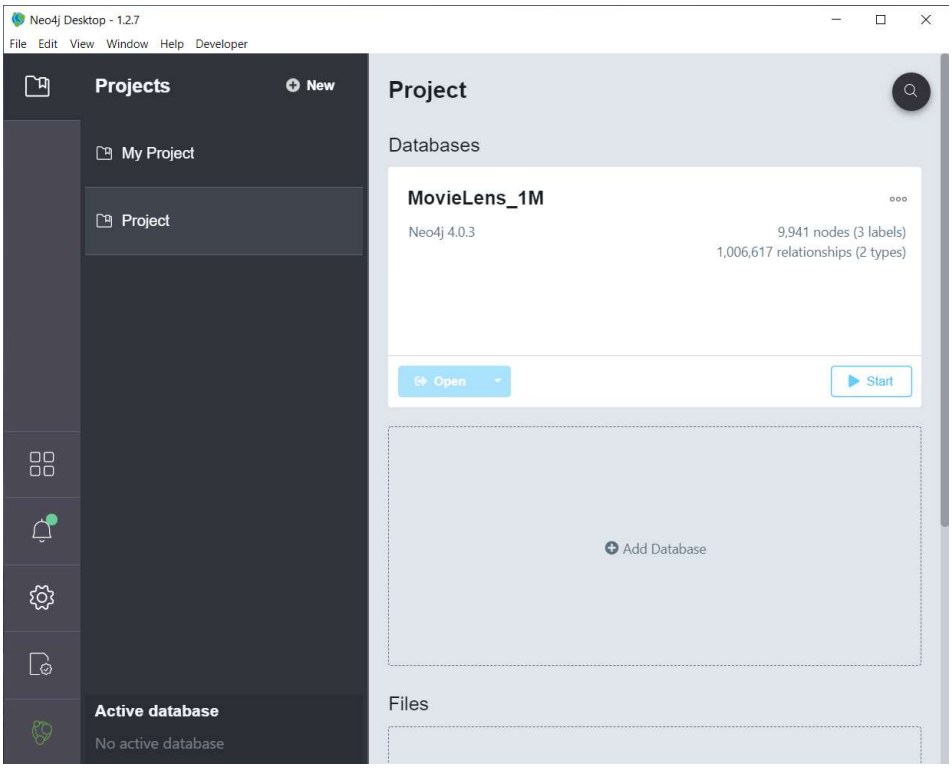
خروجی برنامه

Index	Average Age	State Name
City1	48.55	آبادان
City2	49.19	آمل
City3	48.96	اراک
City4	48.94	اردبیل
City5	48.95	ارومیه
City6	47.94	اسلام‌شهر
City7	48.71	اصفهان
City8	47.98	اهواز
City9	48.67	ایلام
City10	48.73	بابل
City11	48.33	بجنورد
City12	49.18	بروجرد
City13	48.65	بندرعباس
City14	48.30	بوشهر
City15	49.19	بیرجند
City16	48.69	تبریز
City17	49.17	تهران
City18	49.08	خرم آباد
City19	48.91	خمینی‌شهر
City20	48.82	خوی

بخش دوم – مدل سازی داده‌ها با گراف/کار با دیتابیس Neo4j

هدف از این بخش مدل‌سازی داده‌ها با گراف یا به عبارتی کار با دیتابیس Neo4j می‌باشد. برای کار با این بستر شیوه‌های مختلفی وجود دارد که ما برای این منظور از نرم‌افزارهای مربوط به محیط ویندوز آن استفاده نموده‌ایم.

در گام اول از سایت رسمی Neo4j، نسخه مربوط به Desktop آن را دانلود و بر اساس دستورالعمل نصب نمودم. با اجرای این برنامه با محیطی مشابه شکل زیر مواجه خواهیم شد.

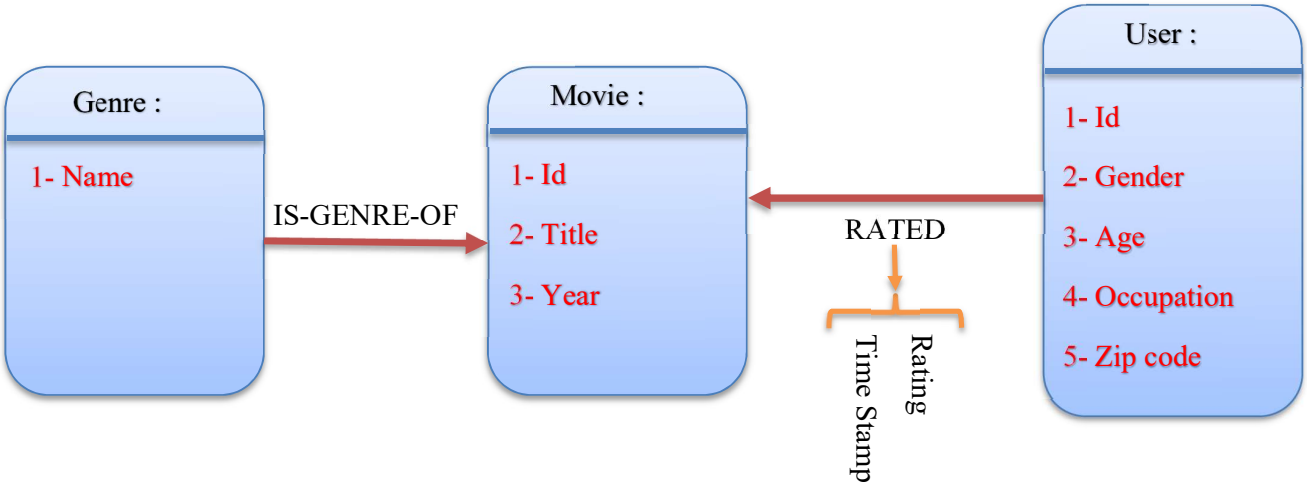


شکل8: محیط گرافیکی کار با دیتابیس Neo4j در ویندوز

در این محیط به راحتی می‌توانیم به ساخت محیط کاری و دیتابیس‌های مختلف بپردازیم و همچنین تمام Queryهای لازم را اجرا نماییم. حال که نرم‌افزار مورد نظر را نصب نمودیم، با ورود به محیط آن یک محیط کاری به نام Project ایجاد می‌نماییم و در گام بعد با فشردن بر روی Add Database در محیط سمت راست که در تصویر بالا نیز قابل روئیت می‌باشد، به ساخت یک دیتابیس می‌پردازیم. ما در اینجا نام دیتابیس را MovieLens_1M می‌گذاریم و کلمه عبور آن را نیز bigdata در نظر می‌گیریم و به این ترتیب دیتابس ما ایجاد می‌شود که در ابتدا هیچ داده‌ای در داخل آن قرار ندارد.

در مرحله بعد بر روی دکمه Start در دیتابیس مورد نظر می‌فشاریم تا اتصال به دیتابیس مهیا شود، حال برای اینکه داده‌های خود را در دیتابیس مورد نظر بارگذاری نماییم از کد آماده‌ایی که در متن تمرین به آن اشاره شده‌بود استفاده می‌نماییم.

در اینجا هدف این است که دیتاست مربوط به یک میلیون فیلم که در MovieLens می‌باشد را در دیتابیس بارگذاری نماییم. این دیتاست از سه فایل مجزا تشکیل شده‌است که فایل اول آن به نام movie.dat ، شامل اطلاعات Movie Id، تیتر فیلم که از آن سال ساخت را نیز می‌توان استخراج نمود و در نهایت ژانرهای مختلف فیلم می‌باشد. فایل دوم User.dat ، شامل User Id، جنسیت کاربر، سن، شغل و کدپستی می‌باشد و در نهایت فایل سوم Rating.dat ، شامل User Id، Id مربوط به فیلمی که کاربر به آن رای داده‌است و در نهایت امتیاز داده شده را در خود جای داده‌است. برای ذخیره این اطلاعات به صورت گرافی ساختار و ارتباط نودها به صورت زیر می‌باشد.



شکل9: ساختار و ارتباط بین نودها در دیتاست MovieLens جهت ذخیره در Neo4j

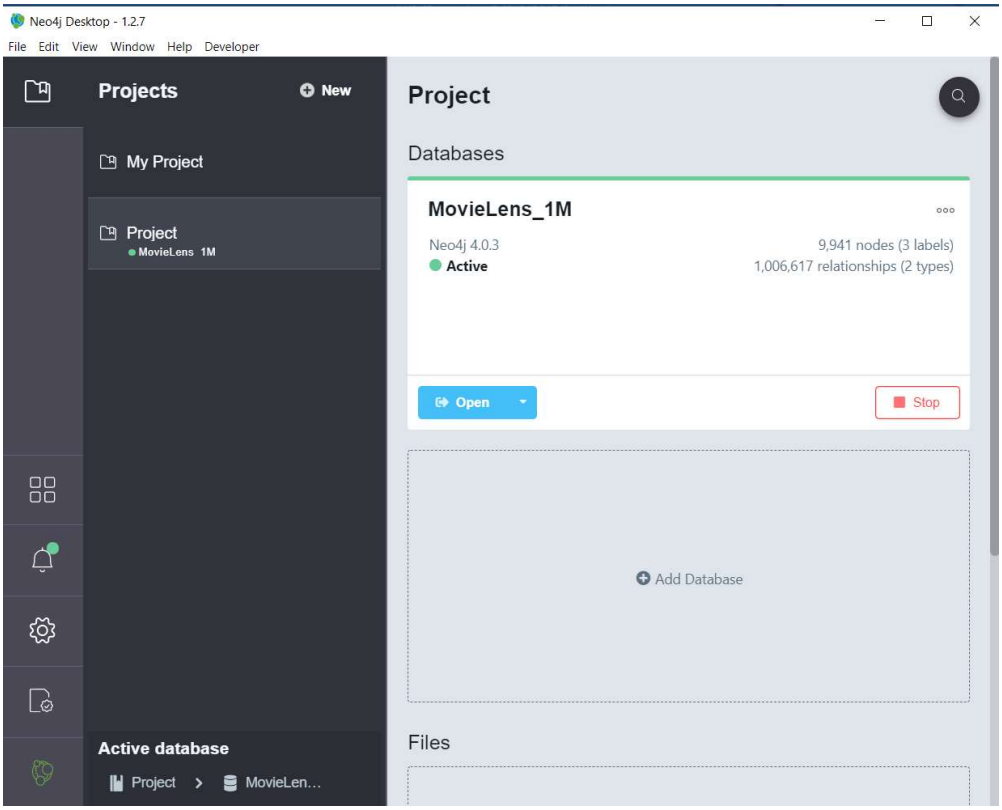
حال برای اینکه ساختار فوق را در دیتابیس ساخته شده در Neo4j ذخیره نماییم از کد مربوط به Load_data.py که در پوشه کدها و در زیرپوشه HW2_Step2 قرار دارد استفاده می‌کنیم. این کد به کمک کتابخانه py2neo که جهت کار با دیتابیس Neo4j می‌باشد به ساخت نودها و بارگذاری داده‌ها می‌پردازد. البته در کد مورد نظر جهت ذخیره داده‌های ساختار بالا، نسبت به کد معرفی شده تغییراتی صورت گرفته است تا بتواند ساختار بالا را پیاده‌سازی نماید. این کد از لحاظ ساختار بهینه نمی‌باشد و به همین علت برای بارگذاری یک میلیون داده در دیتابیس در حدود 2 ساعت و نیم زمان نیاز دارد و روش

بهتر این است که از دستورات Bulk Write و غیره استفاده نماییم ولی در این بخش ما تمرکز خود را بیشتر روی بخش پرس‌وجوها قرار داده‌ایم. یکی از بخش‌های مهم این کد را در زیر مشاهده می‌نمایید.

```
USERNAME = "neo4j"
PASS = "bigdata" #default

graph = Graph("bolt://localhost:7687", auth = (USERNAME, PASS))
```

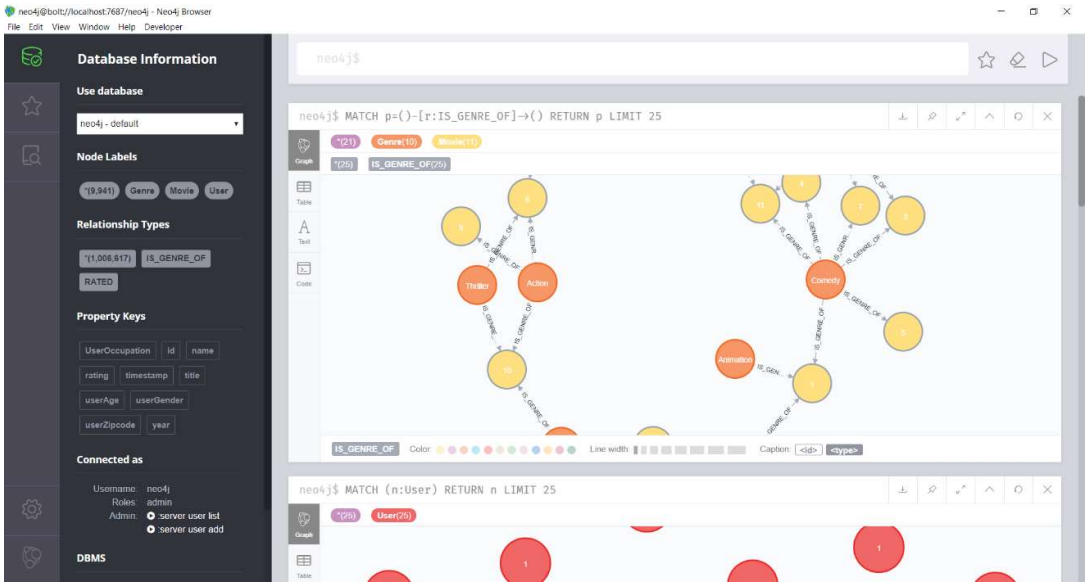
جهت ارتباط با دیتابیس ساخته شده در Neo4j نیاز داریم که به کمک تابع Graph یک Object از دیتابیس ایجاد نماییم که لینک ارتباطی و اطلاعات کاربری را باید در آن وارد نماییم و خروجی آن graph می‌باشد که برای ارتباط و اجرای دستورات در دیتابس از آن استفاده می‌نماییم. حال بعد از بارگذاری دیتاست فوق در دیتابیس، وضعیت دیتابیس به صورت شکل زیر می‌باشد.



شکل10: دیتابیس MovieLens_1M در Neo4j بعد از بارگذاری دیتاست

همانطور که در شکل فوق مشاهده می‌نمایید، دیتابیس فعال می‌باشد و تعداد نودها و ارتباطات موجود در دیتابیس را می‌توانیم مشاهده نماییم. برای اینکه بتوانیم Query های خود را در Neo4j اجرا نماییم و یا اطلاعات و ارتباطات گرافیکی موجود در دیتابیس را مشاهده نماییم، می‌توانیم بر روی گزینه Open در

شکل10 بفشاریم تا وارد محیطی که در شکل 11 مشاهده می‌نمایید، شویم و به جزئیات داده‌ها دسترسی داشته‌باشیم.



شکل11: محیط اجرای درخواست‌ها و مشاهده جزئیات ساختار گراف‌ها و غیره

البته ما برای اجرای درخواست‌ها و مشاهده نتایج آن‌ها که در ادامه توضیح خواهیم داد از این محیط استفاده نمی‌کنیم. تمامی درخواست‌ها در قالب کد پایتون و به کمک کتابخانه py2neo پیاده‌سازی شده‌است که در پوشه کدها و در زیرپوشه HW2_Step2 به تفکیک هر سوال، کد مورد نظر آن نیز آورده شده‌است.

سوالات و پرس‌وجوها

در این قسمت می‌خواهیم نحوه بیان درخواست را با فرمت بکار رفته در Neo4J بیان نماییم. در ادامه با توجه به سوالات مورد نظر، درخواست مورد نظر و خروجی آن را بیان می‌نماییم.

سوال 1)

در این بخش می‌خواهیم بررسی نماییم که چه ژانرهایی در این دیتاست وجود دارد. برای این منظور درخواست زیر را به دیتابیس ارسال می‌نماییم.

```
results = movieLens_Graph.run('MATCH (g:Genre)'
                              'RETURN g.name')
```

در کد بالا که در بخش‌های بعدی نیز آن را مشاهده خواهیم نمود، movieLens_Graph در واقع نمونهٔ ایجاد شده از دیتابیس می‌باشد که به کمک تابع run() در آن می‌توانیم درخواست‌های Neo4j را اجرا نماییم. در درخواست فوق، ابتدا به کمک MATCH تمامی نودهایی که مربوط به ژانر می‌باشد را تعیین نموده‌ایم و سپس مقدار نام آن‌ها را در قالب g.name باز می‌گردانیم. خروجی حاصل از اجرای کد فوق به صورت زیر می‌باشد.

خروجی برنامه

```
There are below genres in this database :
==================================================
1       Action
2       Adventure
3       Animation
4       Children's
5       Comedy
6       Crime
7       Documentary
8       Drama
9       Fantasy
10      Film-Noir
11      Horror
12      Musical
13      Mystery
14      Romance
15      Sci-Fi
16      Thriller
17      War
18      Western
==================================================
Total time for this query : 0.0050 sec
```

همانطور که مشاهده می‌نمایید، زمان اجرای این درخواست برابر 5 میلی‌ثانیه می‌باشد.

سوال 2)

در این قسمت می‌خواهیم تعداد کل فیلم‌های موجود در این دیتابیس را بدست آوریم. برای این منظور درخواست زیر را ارسال می‌نماییم.

```
results = movieLens_Graph.run('MATCH (m:Movie)'
                              'RETURN count(*)')
```

همانطور که در بالا مشاهده می‌نمایید، ابتدا به کمک دستور MATCH که مانند شرط عمل می‌کند، بررسی را بر روی نودهای فیلم اعمال می‌نماییم و سپس تابع count(*) را بازمی‌گردانیم. به این ترتیب به ازای هر نود فیلم یک واحد شمارش صورت می‌گیرد و در نهایت مقدار نهایی تعداد فیلم‌ها بازگردانده می‌شود. در زیر خروجی این بخش را مشاهده می‌نمایید.

خروجی برنامه

```
Total Number of Movies in this database are : 3883
================================================================
Total time for this query : 0.0479 sec
```

همانطور که مشاهده می‌کنید، زمان اجرای این درخواست برابر 47 میلی‌ثانیه می‌باشد. کد مربوط به این بخش در پوشه کدها و در زیرپوشه HW2_Step2 و با نام Part2_Q2.py موجود می‌باشد.

سوال 3)

در این قسمت می‌خواهیم تعداد امتیازهای داده شده در هر رده امتیازی را برای فیلم Silence of the Lambs بدست آوریم. کد مربوط به این بخش در پوشه کدها و در زیر پوشه HW2_Step2 و با نام Part2_Q3.py قابل مشاهده می‌باشد. همانطور که جلوتر مشاهده خواهید نمود، درخواست مربوط به این قسمت برای اجرا به زمانی در حدود 813 میلی‌ثانیه نیاز دارد.

کد مربوط به درخواست مورد نظر را در ادامه مشاهده می‌کنید.

```
results = movieLens_Graph.run(
'MATCH (m:Movie {title:"Silence of the Lambs, The"}) <- [r:RATED {rating:1}] - (u:User)'
'RETURN count(*)'
'UNION ALL '
'MATCH (m:Movie {title:"Silence of the Lambs, The"}) <- [r:RATED {rating:2}] - (u:User)'
'RETURN count(*)'
'UNION ALL '
'MATCH (m:Movie {title:"Silence of the Lambs, The"}) <- [r:RATED {rating:3}] - (u:User)'
'RETURN count(*)'
'UNION ALL '
'MATCH (m:Movie {title:"Silence of the Lambs, The"}) <- [r:RATED {rating:4}] - (u:User)'
'RETURN count(*)'
'UNION ALL '
'MATCH (m:Movie {title:"Silence of the Lambs, The"}) <- [r:RATED {rating:5}] - (u:User)'
'RETURN count(*)')
```

همانطور که در کد بالا مشاهده می‌نمایید، ما به کمک دستور UNION ALL این امکان را که چندین درخواست به صورت همزمان به دیتابیس ارسال شوند و جواب‌های آن‌ها نیز به صورت یکپارچه بازگردانده شوند، مهیا می‌نماییم. در هر قسمت به کمک دستور MATCH ابتدا شرایط مورد نظر را ایجاد می‌کنیم که در اینجا به نود فیلم اشاره داریم که فیلد مربوط به title آن همان اسم فیلم مورد نظر باشد و سپس به کمک خط تیره و پیکان‌ها می‌توانیم ارتباط بین نودها را نشان دهیم که در بالا می‌توان نشان داد که کاربرها به فیلم‌ها امتیاز یا Rating می‌دهند و در هر قسمت برای ارتباط Rated به کمک فیلد rating مقدار امتیاز داده شده را مشخص می‌نماییم و در نهایت به کمک Count(*) تعداد آن را می‌شماریم.

به طور خلاصه خط اول این درخواست می‌گوید تعداد امتیازهای 1 که به فیلم Silence of The Lambs توسط کاربران داده شده‌است، چقدر است و این فرآیند برای دیگر امتیازها نیز تکرار شده‌است. با اجرای این درخواست، خروجی برنامه به صورت زیر می‌باشد.

خروجی برنامه

```
Rating Summery for "Silence of the Lambs" movie :
=================================================================
1.Total Rating with rate = 1     37
2.Total Rating with rate = 2     43
3.Total Rating with rate = 3     246
4.Total Rating with rate = 4     902
5.Total Rating with rate = 5     1350
=================================================================
Total time for this query : 0.8139 sec
```

سوال 4)

در این قسمت می‌خواهیم تعداد امتیازات هر فیلم را بدست آوریم و نتایج را به صورت نزولی مرتب نماییم. برای این منظور درخواست زیر را به دیتابیس ارسال می‌نماییم.

```
results = movieLens_Graph.run(
                        'MATCH (m:Movie) <- [r:RATED] - (u:User)'
                        'WITH m, count(r) as TotalrateForEachMovie '
                        'ORDER BY TotalrateForEachMovie DESC '
                        'RETURN TotalrateForEachMovie, m.title LIMIT 20')
```

همانطور که در کد بالا مشاهده می‌نمایید، ابتدا به کمک MATCH مشخص می‌نماییم که می‌خواهیم پردازش را بر روی چه فیلدهایی انجام دهیم که در اینجا نودهای فیلم و امتیاز که با m و r نمایش داده شده‌است مورد نظر ما می‌باشند. سپس در گام بعدی به کمک WITH مشخص می‌نماییم برای هر فیلم (m) تعداد امتیازات داده شده (count(r)) را بشمارد و در نهایت به کمک ORDER BY و DESC آن‌ها را به صورت نزولی مرتب می‌نماییم. در پایان به دلیل اینکه تعداد فیلم‌ها بسیار زیاد می‌باشد، با اعمال LIMIT 20 فقط بیست فیلد اول خروجی را نمایش می‌دهیم. با حذف این قسمت می‌توانیم تمام فیلم‌ها را مشاهده نماییم. در زیر خروجی این کد را مشاهده می‌نمایید.

خروجی برنامه

```
Number of Rating for Each Movie in Descending Order :
================================================================
Number of rating: 3428   Movie name: "American Beauty"
Number of rating: 2991   Movie name: "Star Wars: Episode IV - A New Hope"
Number of rating: 2990   Movie name: "Star Wars: Episode V - The Empire Strikes Back"
Number of rating: 2883   Movie name: "Star Wars: Episode VI - Return of the Jedi"
Number of rating: 2672   Movie name: "Jurassic Park"
Number of rating: 2653   Movie name: "Saving Private Ryan"
Number of rating: 2649   Movie name: "Terminator 2: Judgment Day"
Number of rating: 2590   Movie name: "Matrix, The"
Number of rating: 2583   Movie name: "Back to the Future"
Number of rating: 2578   Movie name: "Silence of the Lambs, The"
Number of rating: 2538   Movie name: "Men in Black"
Number of rating: 2514   Movie name: "Raiders of the Lost Ark"
Number of rating: 2513   Movie name: "Fargo"
Number of rating: 2459   Movie name: "Sixth Sense, The"
Number of rating: 2443   Movie name: "Braveheart"
Number of rating: 2369   Movie name: "Shakespeare in Love"
Number of rating: 2318   Movie name: "Princess Bride, The"
Number of rating: 2304   Movie name: "Schindler's List"
Number of rating: 2288   Movie name: "L.A. Confidential"
Number of rating: 2278   Movie name: "Groundhog Day"
================================================================
Total time for this query : 1.1005 sec
```

کد مربوط به این قسمت در بخش کدها و در زیرپوشه HW2_Step2 و با نام Part2_Q4.py قابل مشاهده می‌باشد.

سوال 5)

در این قسمت چندین مسئله را می‌خواهیم بررسی نماییم. اول اینکه کدام ژانر بیشترین فیلم تولید شده را دارد و همچنین کدام ژانر بیشترین امتیاز کاربران و کدام ژانر کمترین آن‌ها را دارد. کد مربوط به این درخواست را در زیر مشاهده می‌نمایید.

```
results = movieLens_Graph.run(
        'MATCH (g:Genre) - [igf:IS_GENRE_OF] -> (m:Movie) '
        'WITH g, count(m) as TotalMovieInEachGenre '
        'ORDER BY TotalMovieInEachGenre DESC '
        'RETURN TotalMovieInEachGenre as result, g.name LIMIT 1 '
        'UNION ALL '
        'MATCH (g:Genre) - [igf:IS_GENRE_OF] -> (m:Movie)  <- [r:RATED] - (u:User) '
        'WITH g, avg(r.rating) as AvgRatingInEachGenre '
        'ORDER BY AvgRatingInEachGenre DESC '
        'RETURN AvgRatingInEachGenre as result, g.name LIMIT 1 '
        'UNION ALL '
        'MATCH (g:Genre) - [igf:IS_GENRE_OF] -> (m:Movie)  <- [r:RATED] - (u:User) '
        'WITH g, avg(r.rating) as AvgRatingInEachGenre '
        'ORDER BY AvgRatingInEachGenre '
        'RETURN AvgRatingInEachGenre as result, g.name LIMIT 1')
```

همانطور که در کد بالا مشاهده می‌نمایید، به کمک UNION ALL چندین درخواست همزمان را ارسال نموده‌ایم. در درخواست اول ابتدا به کمک MATCH فیلدهای مورد بررسی یعنی فیلم‌ها و ژانرها را تعیین نموده‌ایم و در ادامه برای هر ژانری تعداد فیلم‌های آن را می‌شماریم و به عنوان TotalMovieInEachGenre در نظر می‌گیرم و در ادامه آن را به صورت نزولی مرتب می‌نماییم و عنصر اول آن را به عنوان ژانر با بیشترین فیلم باز می‌گردانیم.

در درخواست بعدی مجددا به کمک MATCH فیلدهای ژانر و امتیاز را بر اساس ارتباطشان در گراف مشخص می‌نماییم و سپس به کمک دستور WITH و تابع avg() به محاسبه میانگین امتیازات به ازای هر ژانر می‌پردازیم و در ادامه آن‌ها را به صورت نزولی مرتب کرده و عنصر ابتدای آن را به عنوان بیشتر متوسط امتیاز باز می‌گردانیم و همین کار را مجددا برای بدست آوردن مقدار ماکزیمم انجام می‌دهیم.

کد مربوط به این قسمت در بخش کدها و در زیرپوشه HW2_Step2 و با نام Part2_Q5.py قابل مشاهده و اجرا می‌باشد. با اجرای درخواست فوق خروجی برنامه به صورت زیر خواهد بود.

خروجی برنامه

```
The Result of This Question :
======================================================================
The Most Production Movies is in "Drama" Genre and The Total Number Movies are: 1603
The Most Movies Rating is in "Film-Noir" Genre and Its Average Rating is: 4.075188
The Least Movies Rating is in "Horror" Genre and Its Average Rating is: 3.215013
======================================================================
Total time for this query : 7.3151 sec
```

سوال 6)

در این قسمت می‌خواهیم عنوان فیلم‌های تولید شده در سال 2000 را بدست آوریم. برای این کار نیاز به یک پیش‌پردازش بر روی داده‌ها وجود دارد که ما آن را در همان مرحله ذخیره‌سازی داده‌ها در دیتابیس انجام دادیم و سال تولید فیلم را که در عنوان فیلم قرار داشت جدا نموده و به عنوان یه فیلد مجزا به نام year در نودهای مربوط به Movie قرار داده‌ایم و به این ترتیب انجام این درخواست بسیار ساده خواهد بود. در زیر کد مربوط به این درخواست را مشاهده می‌نمایید.

```
results = movieLens_Graph.run('MATCH (m:Movie {year:"2000"})'
                              'RETURN m.title')
```

همانطور که در دستور بالا مشاهده می‌نمایید، بر روی نودهای فیلم که سال تولید آن‌ها برابر 2000 می‌باشد، عملیات MATCH را انجام داده‌ایم و به ازای این نودها مقدار عنوان یا title را به عنوان خروجی باز می‌گردانیم.

با اجرای کد مورد نظر به تعداد 156 عنوان فیلم باز گردانده می‌شود که در ادامه بخشی از آن را به عنوان خروجی برنامه خواهیم دید.

خروجی برنامه

```
146     Girlfight
147     Remember the Titans
148     Bamboozled
149     Bootmen
150     Digimon: The Movie
151     Get Carter
152     Meet the Parents
153     Requiem for a Dream
154     Tigerland
155     Two Family House
156     Contender, The
======================================================================
Total time for this query : 0.0538 sec
```

کد مربوط به این قسمت در بخش کدها و در زیرپوشه HW2_Step2 و با نام Part2_Q6.py قابل مشاهده می‌باشد.

سوال 7)

در این قسمت می‌خواهیم بررسی نماییم که برنامه‌نویسان به کدام فیلم بیشترین امتیاز را داده‌اند و یا کدام فیلم تعداد بیشتری امتیاز 5 را توسط آن‌ها دریافت کرده‌است. برای این منظور در ابتدا باید به این نکته توجه نماییم که در فایل مربوط به کاربران که با نام user.dat موجود می‌باشد، فیلدی به نام Occupation وجود دارد که این فیلد شغل افراد را مشخص می‌نماید. ما این فیلد را در نودهای مربوط به کاربران ذخیره نموده‌ایم و با توجه به توضیحات مربوط به دیتاست مقدار عددی 12 اشاره به برنامه‌نویسان دارد. بنابراین با توجه به این توضیحات درخواست مورد نظر برای این منظور به صورت زیر می‌باشد.

```
results = movieLens_Graph.run(
        'MATCH (u:User {UserOccupation:"12"})-[r:RATED {rating:5}]->(m:Movie) '
        'WITH m, count(r) as rate_five_count '
        'ORDER BY rate_five_count DESC '
        'RETURN rate_five_count, m.title LIMIT 1')
```

همانطور که در کد بالا مشاهده می‌نمایید به کمک دستور MATCH فیلدهای کاربران و امتیاز و ارتباط آن‌ها را مشخص می‌نماییم. نکته حایز اهمیت این است که فیلتری نیز بر روی آن‌ها اعمال نموده‌ایم که فقط کاربرانی را که شغل آن‌ها برنامه‌نویس می‌باشد (UserOccupation:12) و امتیاز 5 (rating:5) را به فیلم‌ها داده‌اند مد نظر ما می‌باشند. سپس در ادامه به ازای تمام فیلم‌هایی که در ساختار MATCH صدق می‌کنند تعداد امتیازات را می‌شماریم و آن‌ها را به کمک ORDER BY به صورت نزولی مرتب می‌نماییم و در انتها به کمک LIMIT 1 فقط عنصر ابتدایی را که بیشترین آن‌ها می‌باشد، به عنوان خروجی باز می‌گردانیم. در ادامه خروجی این قسمت را مشاهده می‌نمایید.

خروجی برنامه

```
The Most five Rating by Programmer:
================================================================
Number of 5-Rating: 155  Movie name: Star Wars: Episode IV - A New Hope
================================================================
Total time for this query : 0.3162 sec
```

کد مربوط به این سوال در بخش کدها و در زیرپوشه HW2_Step2 و با نام Part2_Q7.py قابل مشاهده می‌باشد.

سوال 8)

در این قسمت می‌خواهیم پنج فیلم محبوب رده‌سنی بین 18 تا 34 را بدست آوریم که ملاک ما میانگین امتیازات داده شده به آن‌ها می‌باشد. نکته‌ای که در اینجا وجود دارد این است که هیچ شرطی بر روی تعداد رای‌های داده شده به هر فیلم اعمال نشده‌است و بنابراین اگر تنها یک نفر به فیلمی امتیار 5 بدهد، بنابراین این فیلم به عنوان محبوب‌ترین‌ها در نظر گرفته می‌شود. نکته دیگری که باید به آن توجه نماییم این است که در نودهای مربوط به کاربران فیلدی به نام Age وجود دارد که سن کاربران را نشان می‌دهد ولی با توجه به توضیحات دیتاست این فیلد به صورت بازه‌ای می‌باشد که بر این اساس عدد 18 اشاره به سنین بین 18 تا 24 سال دارد و عدد 25 نمایانگر گروه سنی بین 25 تا 34 سال می‌باشد. حال با توجه به توضیحات داده شده در بالا، درخواست مورد نظر به صورت زیر می‌باشد.

```
results = movieLens_Graph.run(
                        'MATCH (u:User)-[r:RATED]->(m:Movie) '
                        'WHERE (u.userAge = "18" OR u.userAge = "25") '
                        'WITH m, avg(r.rating) as avrage_rating '
                        'ORDER BY avrage_rating DESC '
                        'RETURN avrage_rating, m.title LIMIT 5')
```

در ابتدا به کمک MATCH نودها و ارتباطاتی را که می‌خواهیم مورد بررسی قرار دهیم مشخص می‌نماییم. در اینجا با کاربران، امتیازات آن‌ها و فیلم‌ها سروکار داریم. در خط بعد به کمک WHERE می‌توانیم بر روی سن افراد شرط اعمال نماییم، بنابراین به این ترتیب ما فقط با کاربرانی که محدوده سنی آن‌ها بین 18 تا 34 سال می‌باشد سروکار داریم (userAge با مقادیر 18 و 25 اشاره به این رده سنی دارد). حال در ادامه به کمک WITH متوسط امتیازاتی که این کاربران به هر فیلم داده‌اند را بدست می‌آوریم و در بخش بعد به کمک ORDER BY آنها را به صورت نزولی مرتب می‌نماییم و در نهایت با دستور LIMIT 5 فقط 5 آیتم اول که همان 5 فیلم برتر می‌باشد را باز می‌گردانیم.

کد مربوط به این بخش در پوشه کدها و در زیرپوشه HW2_Step2 و با نام Part2_Q8.py مشخص شده است که با اجرای آن خروجی برنامه به صورت زیر خواهد بود.

خروجی برنامه

```
The Most Favorit Movies by 18-34 Years Old:
======================================================================
1. The Movie Rating: 5.0        Movie Name: Callejón de los milagros, El
2. The Movie Rating: 5.0        Movie Name: Gate of Heavenly Peace, The
3. The Movie Rating: 5.0        Movie Name: Baby, The
4. The Movie Rating: 5.0        Movie Name: Dingo
5. The Movie Rating: 5.0        Movie Name: Black Sunday (La Maschera Del Demonio)
======================================================================
Total time for this query : 1.1360 sec
```

سوال 9)

در این قسمت می‌خواهیم بیست فیم محبوب از لحاظ میانگین امتیازات را بدست آوریم، با این شرط که حداقل 100 نفر به این فیلم‌ها رای داده باشند. این شرط باعث می‌شود که مثلا اگر فیلمی فقط توسط یک نفر امتیاز 5 را دریافت کرده باشد، به عنوان فیلم برتر انتخاب نشود. برای مهیا شدن خواسته فوق، درخواست زیر را اجرا می‌کنیم.

```
results = movieLens_Graph.run(
    'MATCH (u:User)-[r:RATED]->(m:Movie) '
    'WITH m, count(u) as numOfUserRated, avg(r.rating) as avrage_rating '
    'WHERE numOfUserRated > 100 '
    'RETURN avrage_rating, numOfUserRated, m.title '
    'ORDER BY avrage_rating DESC LIMIT 20')
```

مانند تمامی درخواست‌ها، این درخواست نیز با ساختار MATCH شروع می‌شود که به کاربران، امتیاز-های داده شده توسط آن‌ها به فیلم‌ها و ارتباطات بین آن‌ها اشاره دارد. حال در قدم بعدی به کمک WITH و در دو قسمت برای هر فیلم هم تعداد کاربرانی که به آن‌ها امتیاز داده‌اند و هم متوسط امتیازهای داده شده را بدست می‌آوریم و آن‌ها را به ترتیب به numOfUserRated و Avrage_rating نسبت می‌دهیم و در گام بعد به کمک WHERE بر روی تعداد کاربران که در مرحله قبل بدست آمده است شرط اعمال می‌نماییم و فقط مواردی که تعداد کاربران آن از 100 بیشتر باشد را به گام بعد انتقال می‌دهیم و در نهایت با مرتب سازی آن‌ها به صورت نزولی، اطلاعات مربوط به 20 فیلم اول را به کمک LIMIT 20 به خروجی ارسال می‌نماییم.

کد مربوط به این قسمت در بخش کدها و در زیرپوشه HW2_Step2 و با نام Part2_Q9.py قابل مشاهده می‌باشد. با اجرای این کد، خروجی آن را در ادامه مشاهده می‌نمایید.

خروجی برنامه

```
Top 20 Movies by Average User Rating:
==============================================================
1       Number of user rating:628       Average rating:4.560510         Movie Name: Seven Samurai (The Magnificent Seven) (Shichinin no samurai)
2       Number of user rating:2227      Average rating:4.554558         Movie Name: Shawshank Redemption, The
3       Number of user rating:2223      Average rating:4.524966         Movie Name: Godfather, The
4       Number of user rating:657       Average rating:4.520548         Movie Name: Close Shave, A
5       Number of user rating:1783      Average rating:4.517106         Movie Name: Usual Suspects, The
6       Number of user rating:2304      Average rating:4.510417         Movie Name: Schindler's List
7       Number of user rating:882       Average rating:4.507937         Movie Name: Wrong Trousers, The
8       Number of user rating:470       Average rating:4.491489         Movie Name: Sunset Blvd. (a.k.a. Sunset Boulevard)
9       Number of user rating:2514      Average rating:4.477725         Movie Name: Raiders of the Lost Ark
10      Number of user rating:1050      Average rating:4.476190         Movie Name: Rear Window
11      Number of user rating:230       Average rating:4.473913         Movie Name: Paths of Glory
12      Number of user rating:2991      Average rating:4.453694         Movie Name: Star Wars: Episode IV - A New Hope
13      Number of user rating:480       Average rating:4.452083         Movie Name: Third Man, The
14      Number of user rating:1367      Average rating:4.449890         Movie Name: Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb
15      Number of user rating:438       Average rating:4.426941         Movie Name: Wallace & Gromit: The Best of Aardman Animation
16      Number of user rating:928       Average rating:4.425647         Movie Name: To Kill a Mockingbird
17      Number of user rating:551       Average rating:4.415608         Movie Name: Double Indemnity
18      Number of user rating:1669      Average rating:4.412822         Movie Name: Casablanca
19      Number of user rating:2459      Average rating:4.406263         Movie Name: Sixth Sense, The
20      Number of user rating:215       Average rating:4.404651         Movie Name: Yojimbo
==============================================================
Total time for this query : 1.2711 sec
```

بخش سوم – دیتابیس‌های سطح گسترده/کار با HBase

در این قسمت می‌خواهیم درباره کار با دیتابیس‌ها سطح گسترده از جمله HBase صحبت نماییم. همانطور که می‌دانیم این دیتابیس‌ها برای ذخیره داده‌های حجیم بر روی ساختارهای توزیع‌شده طراحی شده‌اند و معمولا مبتنی بر کوئری می‌باشند، به همین دلیل ما در ادامه نیاز به طراحی جداول مختلفی داریم که بتوانیم بر اساس آن‌ها به کوئری‌های موجود در سوال به راحتی پاسخ دهیم.

گام اول – ساخت جداول

در این بخش می‌خواهیم به تفصیل درباره نحوه ساخت جداول مختلف مورد نیاز در این قسمت بپردازیم. قبل از پرداختن به جزئیات هر کدام از جداول ابتدا به صورت کلی درباره نحوه ساخت جداول در HBase صحبت خواهیم کرد.

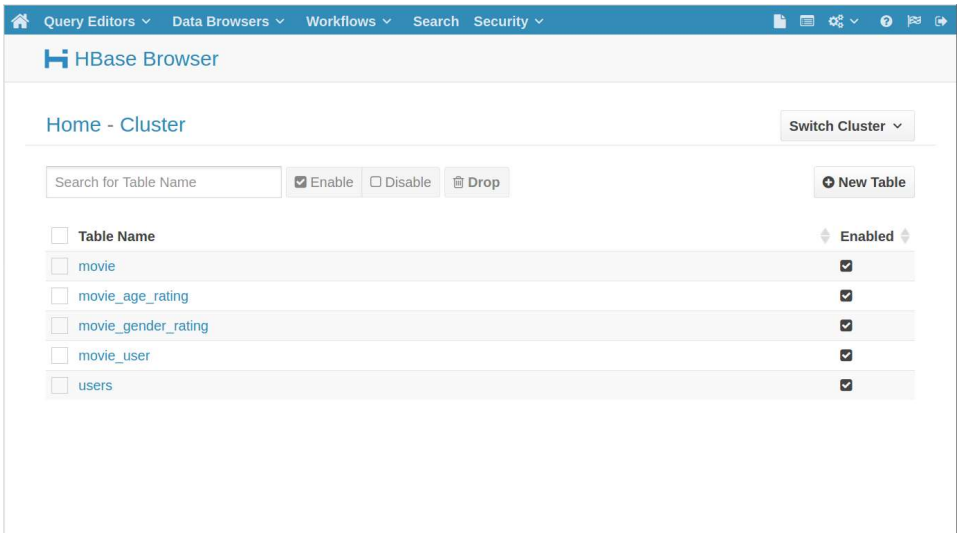
برای کار با دیتابیس HBase روش‌ها مختلفی وجود دارد که یکی از این روش‌ها استفاده از Image مربوط به Cloudera-quichstart در بستر Docker می‌باشد. پیش‌تر و در تمرین قبلی با نحوه راه‌اندازی این بستر به تفصیل آشنا شدیم و در اینجا به جزئیات آن نمی‌پردازیم، تنها نکته‌ای که وجود دارد این است که در صورتی که بخواهیم از کتابخانه Happybase برای ارتباط با HBase استفاده نماییم، نیاز داریم تا پورت 9090 که مربوط به این دیتابیس می‌باشد را فعال نماییم که برای این کار در هنگام ایجاد یک Container از Image مربوطه باید این شمار پورت را نیز با اضافه نمودن دستور p– port_num:port_num فعال نماییم و بعد از این کار به راحتی می‌توانیم با این دیتابیس و به کمک کتابخانه مورد نظر ارتباط برقرار نماییم.

بعد از اجرای Cloudera در داکر، شیوه‌های ارتباطی با دیتابیس HBase به شکل‌های مختلف وجود دارد. اولین و ساده‌ترین راه آن استفاده از بستر گرافیکی Hue می‌باشد. با این محیط پیش‌تر آشنا شده‌ایم و با ورود به آن امکانات مختلفی جهت کار با CDH وجود دارد که یکی از آن‌ها محیط مربوط به HBase می‌باشد. برای ورود به این بستر گرافیکی کافیست با وارد کردن آدرس زیر در یک مرورگر دلخواه و سپس با وارد کردن نام کاربری و کلمه عبور که به صورت پیش‌فرض برابر cloudera می‌باشد، به راحتی به آن وارد شوید.

```
http://localhost:8888
```

برای ارتباط و استفاده از امکانات مربوط به HBase از قسمت نوار ابزار بالای این محیط و با انتخاب Data Browsers به راحتی می‌توانیم گزینه HBase را مشاهده و انتخاب نماییم که با انتخاب آن وارد

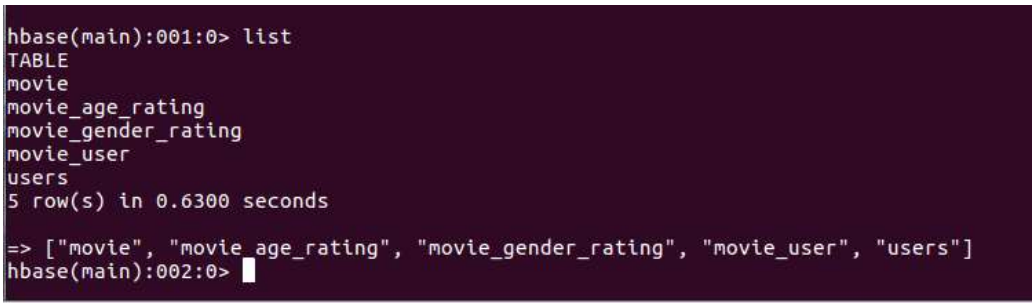
محیط گرافیکی HBase خواهیم شد که در شکل زیر شمایی کلی از این محیط گرافیکی را مشاهده می‌نمایید.



شکل12: شمایی کلی از محیط گرافیکی HBase در بستر Hue

همانطور که در بالا مشاهده می‌نمایید، این بخش مربوط به لیست جداول و ایجاد آن‌ها می‌باشد که کمی جلوتر به تفصیل درباره آن توضیح خواهیم داد.

روش دیگری که برای ارتباط با HBase و ارسال درخواست‌ها می‌توانیم استفاده نماییم محیط Shell مربوط به HBase می‌باشد. برای ورود به این محیط کافیست که در محیط Terminal و بعد از اجرای Container مربوط، دستور hbase shell را وارد نماییم. با اجرای این دستور بعد از مدت زمان اندکی وارد محیط Shell در همان Terminal خواهیم شد. در این محیط به کمک دستورات HBase می‌توانیم به ساخت جدول، ورود داده به جدول و ارسال درخواست‌های مختلف بپردازیم.به عنوان نمونه در شکل زیر با اعمال دستور list، تمامی جداول موجود در HBase را مشاهده می‌نمایید.

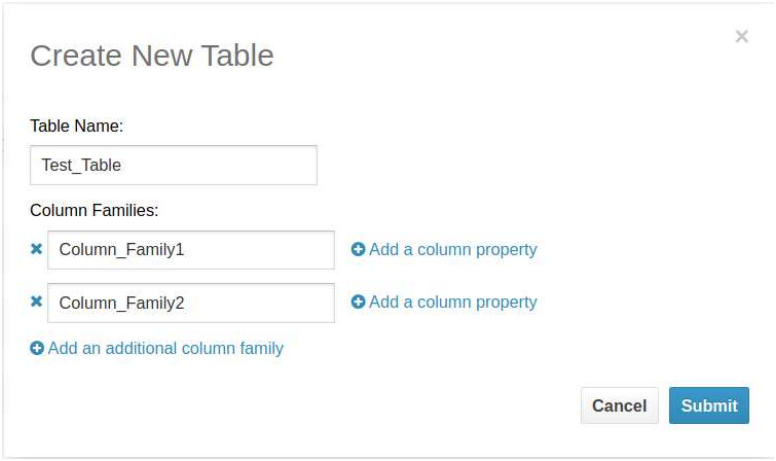


شکل13: محیط Shell مربوط به HBase در Terminal

روش دیگری که می‌توان از آن استفاده نمود، استفاده از کتابخانه Happybase می‌باشد که برای کار با دیتابیس HBase ایجاد شده‌است و تمامی امکانات لازم جهت کار با این دیتابیس را در خود جای داده‌است.

ما در این تمرین برای ساخت جداول و ورود داده‌ها از محیط گرافیکی Hue و برای ارسال درخواست‌ها و پردازش آن‌ها از کتابخانه Happybase در پایتون استفاده نموده‌ایم. در ادامه با این بخش‌ها بیشتر آشنا خواهیم شد. لازم به ذکر است که معمولا برای ورود اطلاعات حجیم به این طریق عمل نمی‌شود و معمولا به کمک کدهای نوشته شده در قالب Map/Reduce به ورود اطلاعات می‌پردازند، ولی در اینجا از آنجایی که ما می‌خواهیم یک تعداد محدودی داده را جهت گرفتن خروجی از کوئری‌ها وارد نماییم، بنابراین از این محیط گرافیکی استفاده نموده‌ایم.

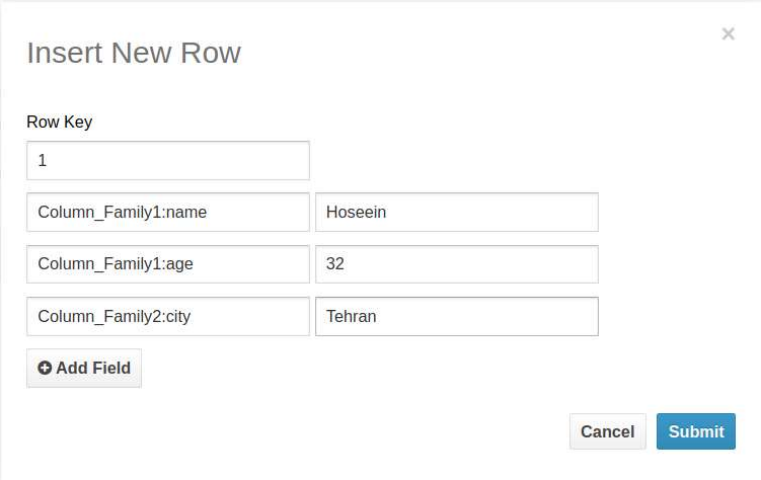
در این قسمت می‌خواهیم به صورت کلی با ساخت یک جدول نمونه و ورود اطلاعات به آن آشنا شویم. برای ساخت جدول بعد از ورود به محیط گرافیکی آن در Hue، می‌توانیم گزینه New Table را انتخاب نماییم. با انتخاب این گزینه به پنجره زیر مواجه خواهیم شد.



شکل14: پنجره مربوط به ساخت جدول در محیط گرافیکی Hue

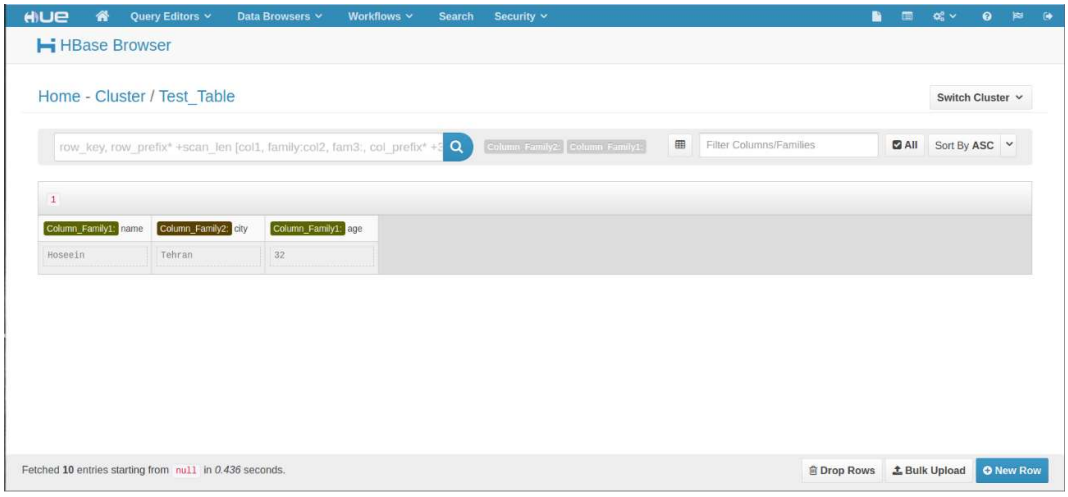
همانطور که در شکل بالا مشاهده می‌نمایید، ابتدا نامی برای جدول انتخاب می‌نماییم و در ادامه به انتخاب Column Familyهای مختلف مورد نیاز در جدول می‌پردازیم که می‌توانیم با فعال نمودن Column Property به اضافه نمودن ویژگی‌های مختلف به این Column Familyها بپردازیم. حال با انتخاب گزینه Submit جدول اولیه ساخته می‌شود. در ادامه با انتخاب جدول مورد نظر از لیست جداول می‌توانیم داده‌های خود را به آن وارد نماییم که در ادامه با محیط آن بیشتر آشنا می‌شویم.

در محیط جدید با انتخاب گزینه New Row در پایین صفحه، اقدام به ورود اطلاعات می‌نماییم. با انتخاب این گزینه با محیط زیر مواجه می‌شویم.



شکل15: پنجره مربوط به ورود اطلاعات مربوط به ستون‌های جدول طراحی شده در محیط Hue

همانطور که در شکل فوق مشاهده می‌نمایید، در قسمت Row Key، کلید مربوط به هر سطر از جدول را وارد می‌نماییم و سپس با توجه به تعریف اولیه از Column Familyهای مختلفی که در ابتدا انجام دادیم، داده‌های مربوط به هر ستون را مانند ساختار بالا به صورت کلید و مقدار وارد می‌نماییم. مثلا در شکل بالا Column_Family1 از ستون‌های نام و سن تشکیل شده‌است. به این ترتیب داده‌ها را وارد کرده و در نهایت با فشردن گزینه Submit این اطلاعات را ثبت می‌نماییم که در نهایت شمای کلی این جدول به صورت زیر می‌باشد.



شکل16: شمای کلی از جدول ساخته‌شده و داده‌های وارد شده در جدول

حال که با نحوه ساخت جداول و وارد کردن داده به آن‌ها به صورت دستی آشنا شدیم، در ادامه با ساختار جداول ساخته شده در این بخش از تمرین جهت پاسخ به کوئری‌های قسمت بعد آشنا خواهیم شد.

جدول مربوط به سوال 1 و 2

برای پاسخ به کوئری‌های مربوط به سوال 1 و 2 در قسمت بعد، جدولی به نام Movie را ایجاد نموده‌ایم. ساختار این جدول بسیار ساده می‌باشد که کلیت آن را در زیر مشاهده می‌نمایید.

جدول1: ساختار کلی جدول مربوط به Movie برای پاسخ به کوئری‌های سوال 1 و 2

<table>
<tr><th colspan="3">Table Name : Movie</th></tr>
<tr><td rowspan="2">Row Key: Movie Id</td><td>Info:</td><td>Genre:</td></tr>
<tr><td><table><tr><td>title</td><td>…</td></tr><tr><td>year</td><td>…</td></tr><tr><td>Num_of_genre</td><td>…</td></tr></table></td><td><table><tr><td>Comedy</td><td>…</td></tr><tr><td>…</td><td>…</td></tr><tr><td>Action</td><td>…</td></tr></table></td></tr>
</table>

همانطور که در جدول بالا مشاهده می‌نمایید، نام این جدول Movie می‌باشد که این جدول از دو Column Family با نام‌های info و genre تشکیل شده‌است. هر سطر این جدول به کمک Movie Id مشخص می‌شود. در Column Family مربوط به Info اطلاعات پایه درباره فیلم شامل نام فیلم، سال ساخت و تعداد ژانرهای هر فیلم وجود دارد و در Column Family مربوط به Genre، اطلاعات مربوط به Genreهای فیلم وجود دارد. به این ترتیب به راحتی می‌توانیم با اعمال فیلتر بر روی ژانرها، به پیدا کردن فیلم‌های یک ژانر خاص بپردازیم و همچین تعداد ژانرهای هر فیلم نیز به راحتی در دسترس و قابل بروزرسانی می‌باشد.

جدول مربوط به سوال 3

در این قسمت می‌خواهیم جدولی طراحی نماییم که به راحتی بتوانیم تعداد افرادی که به هر فیلم رای داده‌اند را بدست آوریم. برای این منظور جدولی به نام movie-user را ایجاد نموده‌ایم که ساختار این جدول را در ادامه مشاهده می‌نمایید.

جدول2: ساختار کلی جدول movie-user مربوط به پاسخ به سوال سوم

<table>
<tr><th colspan="3">Table Name : movie_user</th></tr>
<tr><td rowspan="2">Row Key: Movie Id</td><td>Info:</td><td>users:</td></tr>
<tr><td>
<table>
<tr><td>title</td><td>…</td></tr>
<tr><td>year</td><td>…</td></tr>
</table>
</td><td>
<table>
<tr><td>User Id1</td><td>Rating1</td></tr>
<tr><td>User Id2</td><td>Rating2</td></tr>
<tr><td>…</td><td>…</td></tr>
<tr><td>User Idn</td><td>Ratingn</td></tr>
</table>
</td></tr>
</table>

همانطور که در بالا مشاهده می‌نمایید، ساختار این جدول به این صورت است که کلید هر سطر با Movie Id تعیین می‌شود که به یک فیلم خاص اشاره دارد. این جدول از دو Column Family تشکیل شده‌است. Column Family اول مربوط به اطلاعات پایه فیلم شامل نام فیلم و سال ساخت آن می‌باشد و Column Family دوم مربوط به امتیازاتی می‌باشد که کاربران به این فیلم داده‌اند که نام آن users می‌باشد و ستون‌های آن به ترتیب Id مربوط به هر کاربر و امتیازی که به آن فیلم داده‌است را مشخص می‌نماید.

با ساختار جدول فوق به راحتی می‌توان با یک کوئری اطلاعات مربوط به کاربرانی که به هر فیلم رای داده‌اند را بدست آورد و در نهایت به صورت آفلاین و بدون درگیری شبکه به محاسبه تعداد این افراد برای هر فیلم پرداخت و در نهایت مقدار ماکزیمم آن را گزارش نمود.

نکته‌ای که باید درباره محاسبه آماری در دیتابیس HBase اشاره نماییم، این است که ما معمولا سعی می‌نماییم جداول را به گونه‌ای طراحی نماییم که با یک کوئری تمامی اطلاعات لازم جهت محاسبات آماری را در اختیار داشته‌باشیم و سپس به صورت آفلاین به محاسبات آماری مثل میانگین‌گیری، ماکزیمم و غیره بپردازیم. همچنین می‌توانیم یک جدول جداگانه برای این اطلاعات آماری در نظر بگیریم که آن‌ها را در آن ذخیره نماییم که در هنگام نیاز با یک درخواست بلافاصله به آن‌ها دسترسی داشته باشیم و نیاز به محاسبات مجدد نباشد. البته ما در اینجا این کار را انجام نداده‌ایم ولی شیوه مناسبتر و دسترسی بلادرنگ به این اطلاعات به این صورت می‌باشد.

جدول مربوط به سوال 4

در سوال چهارم می‌خواهیم 3 فیلم و ژانر محبوب رده‌سنی 45 سال به بالا را بدست آوریم. اطلاعات مربوط به دیتاست MovieLens به صورتی می‌باشد که رده‌های سنی را بر اساس اعدادی مشخص به گروه‌ها مختلف تقسیم نموده‌است که این اعداد به صورت جدول زیر می‌باشد.

جدول3: رده‌های سنی مختلف موجود در دیتاست MovieLens

نام گروه	عدد تخصیص داده‌شده	رده‌سنی
Group1	1	زیر 18 سال
Group2	18	از 18 تا 24 سال
Group3	25	از 25 تا 34 سال
Group4	35	از 35 تا 44 سال
Group5	45	از 45 تا 49 سال
Group6	50	از 50 تا 55 سال
Group7	56	56 سال و بیشتر

این اطلاعات به صورت کامل در فایل مربوط به کاربران وجود دارد. حال ما برای سادگی در انجا درخواست مربوط به این بخش به ساخت یک جدول بزرگ می‌پردازیم. ساختار کلی این جدول را در زیر مشاهده می‌نمایید.

جدول4: ساختار کلی جدول movie_age_rating جهت پاسخ به درخواست سوال چهارم

Table Name : movie_age_rating								
Row Key: Movie Id	Info:	Group1:	Group2:	Group3:	Group4:	Group5:	Group6:	Group7:
	title … year … genre …	age 1 User1 … … … User2 …	…	…	…	…	…	age 56 User1 … … … User2 …

همانطور که در جدول movie_age_rating مشاهده می‌نمایید، هر سطر این جدول به کمک Movie_id مشخص می‌شود که اشاره به یک فیلم دارد و این جدول از هشت Column Family متفاوت تشکیل شده‌است. اولین Column Family، اطلاعات پایه‌ای از فیلم شامل نام فیلم، سال ساخت و ژانر فیلم را مشخص می‌نماید. از آنجایی که ما برای کاربران رده‌سنی آن‌ها را می‌دانیم، بنابراین برای هر گروه سنی یک Column Family ایجاد نموده‌ایم که با این ساختار هر یک از کاربران را که به فیلمی رای داده‌اند در گروه سنی خودشان اطلاعاتشان را وارد نموده‌ایم و به این ترتیب به راحتی با یک درخواست و اعمال فیلتر بر روی گروه سنی می‌توانیم فیلم‌هایی که افراد 45 سال به بالا به آن‌ها رای داده‌اند را بدست آوریم. نکته‌ای که وجود دارد این است که لزوما تمامی این Column Familyها برای تمام فیلم‌ها موجود نمی‌باشد، مثلا ممکن است فیلمی فقط شامل امتیازات رده‌سنی گروه اول باشد. بنابراین با اعمال فیلتر بر روی گروه‌های سنی این فیلم به عنوان خروجی بازگردانده نمی‌شود.

جدول مربوط به سوال 5

در این قسمت می‌خواهیم میانگین امتیازاتی که خانم‌ها و آقایان به فیلم‌های ساخته شده در سال 1995 داده‌اند را بدست آوریم و آن‌ها را به تفکیک جنسیت اعلام نماییم. برای این منظور جدولی به نام movie_gender_rating ایجاد نموده‌ایم که ساختار آن به صورت زیر می‌باشد.

جدول5: ساختار کلی جدول movie_gender_rating جهت پاسخ به درخواست سوال پنجم

<table>
<tr><th colspan="4">Table Name : movie_gender_rating</th></tr>
<tr><td rowspan="2">Row Key:
Movie Id</td><td>Info:</td><td>Male:</td><td>Female:</td></tr>
<tr><td><table><tr><td>title</td><td>…</td></tr><tr><td>year</td><td>…</td></tr></table></td><td><table><tr><td>User Id1</td><td>Rating1</td></tr><tr><td>User Id2</td><td>Rating2</td></tr><tr><td>…</td><td>…</td></tr><tr><td>User Idn</td><td>Ratingn</td></tr></table></td><td><table><tr><td>User Id1</td><td>Rating1</td></tr><tr><td>User Id2</td><td>Rating2</td></tr><tr><td>…</td><td>…</td></tr><tr><td>User Idn</td><td>Ratingn</td></tr></table></td></tr>
</table>

همانطور که در جدول فوق مشاهده می‌نمایید، هر سطر این جدول به کمک Movie Id که اشاره به یک فیلم دارد، مشخص می‌شود. این جدول از سه Column Family مختلف تشکیل شده‌است. در Column Family اول اطلاعات پایه فیلم، شامل نام فیلم و سال ساخت آن قرار دارد. از آنجایی که اطلاعات

مربوط به جنسیت کاربران موجود می‌باشد، بنابرین دو Column Family مجزا بر اساس جنسیت افراد ایجاد می‌نماییم و امتیازات کاربران را بر اساس جنسیت آن‌ها در هر یک از ستون‌ها قرار می‌دهیم. با این ساختار به راحتی می‌توانیم با یک درخواست و اعمال فیلتر بر روی سال ساخت فیلم، امتیازات آقایان و خانم‌ها را به تفکیک بدست آوریم و سپس به صورت آفلاین متوسط این امتیازات را محاسبه نماییم.

تا به اینجای کار به تفصیل با ساختار جداول بکار رفته در این بخش از تمرین آشنا شدیم، حال در قسمت بعد با توجه به ساختارهای ذکر شده، به بیان کوئری‌های هر بخش و گزارش خروجی آن‌ها می‌پردازیم.

گام دوم– پرس‌وجوها

در این بخش به کمک کتابخانه آماده Happybase در پایتون به ارتباط با دیتابیس HBase پرداخته‌ایم و کوئری‌های مورد نظر را اعمال نموده‌ایم. برای ارتباط با دیتابیس نیاز به ساخت یک کانکشن می‌باشد که قطعه کد زیر به راحتی این کانکشن را ایجاد می‌نماید.

```
connection = happybase.Connection(host='localhost', port=9090, autoconnect=True)
movie_table = connection.table('movie')
```

همانطور که در کد بالا مشاهده می‌نمایید، با دستور connection و اعمال پارامترهای مورد نیاز، ارتباط خود را با دیتابیس مورد نظر برقرار می‌کنیم و سپس به کمک دستور بعدی و اعمال نام جدول به عنوان ورودی آن، یک نمونه از جدول مورد نظر را ایجاد کرده و به کمک آن به اعمال کوئری‌های مختلف می‌پردازیم. با بیان این مقدمات درباره نحوه اتصال به دیتابیس HBase و جداول، به بیان کوئری‌های هر بخش و خروجی‌های آن می‌پردازیم.

سوال 1)

در این سوال می‌خواهیم به بیان فهرست فیلم‌های یک ژانر مشخص بپردازیم. برای این منظور از جدول movie استفاده می‌نماییم و تابعی به نام movie_filter_based_on_genre() نوشته‌ایم که جدول و ژانر مورد نظر را می‌گیرد و به کمک کوئری زیر به فیلتر داده‌ها می‌پردازد و خروجی مورد نظر را باز می‌گرداند.

```
def movie_filter_based_on_genre(table, genre):
    results = list(table.scan(columns = ['info:title', 'genre:'+genre],
                  filter="SingleColumnValueFilter('genre', '"+genre+"', =,
                                        'binary:"+genre+"', true, false)"))

    return results
```

همانطور که در کوئری بالا مشاهده می‌نمایید، از تابع Scan استفاده شده است که بر روی جدول movie اعمال می‌شود و درخواست شده که فقط داده‌های مربوط به عنوان فیلم و ژانر مورد نظر را بازگرداند و برای اینکه خروجی مناسب حاصل شود از فیلتری به نام SingleCoulmnValueFilter استفاده نموده‌ایم. این فیلتر نام Column Family را به عنوان آرگومان اول و سپس Qualifier را به عنوان آرگومان دوم و سپس به کمک Operand مشخص می‌نماییم که عمل مقایسه بر چه اساس باشد و در نهایت مقدار مورد مقایسه را تعیین می‌نماییم و در انتها با مقادیر Boolean مورد نظر تعیین می‌نماییم که در صورتی که مقدار مورد نظر در سطری وجود ندارد، آن را باز نگرداند و به این ترتیب فقط بر روی سطرهایی که شامل این ستون می‌باشد فیلتر اعمال می‌شود.

با اعمال کوئری مورد نظر بر روی یک نمونه کوچک از جدول که شامل اطلاعات پانزده فیلم مختلف می‌باشد، خروجی را در ادامه مشاهده می‌نمایید.

```
Genre          Movie Title
=====          ===========
Action         GoldenEye
Action         Someone to Watch Over Me
Action         Heat
Action         Sudden Death
```

در خروجی فوق اعمال فیلتر بر روی ژانر Action صورت گرفته‌است که با تغییر آن به فیلم‌های مثلا Drama خروجی به صورت زیر می‌باشد.

```
Genre          Movie Title
=====          ===========
Drama          Dancer in the Dark
Drama          Two Family House
Drama          Waiting to Exhale
```

سوال 2)

در این بخش می‌خواهیم اطلاعات مربوط به تعداد ژانرهای هر فیلم را بدست آوریم. با توجه به اینکه ما این اطلاعات را در قالب num_of_genre در Column Family مربوط به info در جدول Movie قرار داده‌ایم، بنابراین به راحتی و به کمک کوئری زیر می‌توانیم به آن‌ها دسترسی داشته باشیم.

```
# Query
results = list(movie_table.scan(columns = ['info:title', 'info:num_of_genre']))
```

برای این منظور به راحتی با درخواست ستون‌های مربوط به عنوان فیلم و تعداد ژانر آن‌ها می‌توان به این درخواست پاسخ داد که خروجی آن را در نمونه جدول با اطلاعات پانزده فیلم مختلف را در زیر مشاهده می‌نمایید.

Number Of Genres	Movie Title
3	Toy Story
3	GoldenEye
3	Jumanji
3	Someone to Watch Over Me
2	Grumpier Old Men
2	Suture
2	Dancer in the Dark
2	Invisible Man, The
1	Two Family House
2	Waiting to Exhale
1	Father of the Bride Part II
3	Heat
2	Sabrina
2	Tom and Huck
1	Sudden Death

سوال 3)

در این قسمت می‌خواهیم، فیلمی که بیشترین تعداد امتیازدهنده را دارد، بدست آوریم. برای این منظور از جدول movie_user استفاده می‌نماییم و به کمک کوئری زیر تمام اطلاعات لازم برای محاسبه مقدار ماکزیمم تعداد رای‌دهندگان را بدست می‌آوریم.

```
#query
user_rating_list = list(movie_users_table.scan(columns=["info:title", "user"]))
```

همانطور که مشاهده می‌نمایید، به کمک کوئری فوق، اطلاعات مربوط به کاربرانی که به هر فیلم رای داده‌اند بدست می‌آید و سپس به کمک کد پایتون تعداد آن‌ها را به ازای هر فیلم می‌شماریم و ماکزیمم آن‌ها را گزارش می‌نماییم که خروجی این قسمت را در زیر مشاهده می‌نمایید.

```
Movie Title:  Fletch
number of user rating:  14
```

سوال 4)

در این بخش می‌خواهیم سه فیلم و ژانر محبوب مربوط به رده‌سنی بالای 45 سال را بدست آوریم. برای این منظور از جدول movie_age_rating استفاده می‌نماییم. این جدول اطلاعات امتیاز کاربران را به تفکیک گروه سنی ذخیره نموده‌است و به این ترتیب می‌توانیم با اعمال یک فیلتر بر روی رده‌سنی به راحتی فیلم‌ها و ژانرهایی که این رده‌سنی به آن‌ها رای داده‌است را بدست آوریم. کوئری زیر این کار را انجام می‌دهد.

```
#queries for reading movie rating by people who are above 45
result = list(mar_table.scan(
columns = ["info:title", "info:genre", 'group5', 'group6', 'group7'],
filter="SingleColumnValueFilter('group5', 'age', =, 'binary:45', true, false) OR "+
       "SingleColumnValueFilter('group6', 'age', =, 'binary:50', true, false) OR "+
       "SingleColumnValueFilter('group7', 'age', =, 'binary:56', true, false)",
sorted_columns=True))
```

همانطور که مشاهده می‌نمایید، به کمک کوئری بالا، فقط فیلم‌هایی که این گروه سنی به آن‌ها رای داده‌است بازگردانده می‌شود و فقط اطلاعات مربوط به عنوان فیلم، ژانرهای فیلم و اطلاعات امتیازهای داده شده توسط این رده سنی بازگردانده می‌شود و در ادامه به کمک کد پایتون متوسط امتیارها بدست می‌آید و سه فیلم و ژانر برتر آن گزارش می‌شود که خروجی این کد بر روی یک نمونه کوچک از جدول movie_age_rating با 7 عنوان فیلم مختلف به صورت زیر می‌باشد.

```
                                Top 3 Movies
================================================================================

  Average Rating                        Movie Title  (Genre)
------------------      ------------------------------------------------------
        4.50              Lawnmower Man 2: Beyond Cyberspace  ( Sci-Fi|Thriller )
        4.25              Feast of July  ( Drama )
        4.00              Heaven & Earth  ( Action|Drama|War )
```

سوال 5)

آخرین قسمت از این تمرین مربوط به گزارش متوسط امتیازات آقایان و خانم‌ها به فیلم‌های تولید شده در سال 1995 می‌باشد. برای این منظور از جدول movie_gender_rating استفاده شده‌است که برای دریافت اطلاعات مورد نیاز توسط آن از کوئری زیر استفاده شده‌است.

```
#queries for reading male and female rating in 1995 movies
male_result   = list(mgr_table.scan(
  columns=["info:year", "male"], filter="SingleColumnValueFilter('info', 'year',
                                        =, 'binary:1995', true, false)"))
female_result = list(mgr_table.scan(
  columns=["info:year", "female"], filter="SingleColumnValueFilter('info', 'year',
                                         =, 'binary:1995', true, false)"))
```

برای راحتی در محاسبات از دو کوئری مختلف استفاده شده‌است که اطلاعات مربوط به آقایان و خانم‌ها را به تفکیک فقط برای فیلم‌های ساخته شده در سال 1995 بازمی‌گرداند و به این ترتیب به کمک کد پایتون به محاسبه میانگین امتیازات هر گروه جنسی می‌پردازیم و آن را گزارش می‌نماییم. در زیر خروجی این قسمت را مشاهده می‌نمایید.

Gender	Avg Rating for Movies in 1995
Male	2.810
Female	3.267

در پایان لازم به ذکر است که تمامی کدهای مربوط به این قسمت در پوشه کدها و در زیرپوشه HW2_Step3 و به تفکیک قابل دسترس می‌باشد.