

به نام خدا



دانشگاه تهران
پردیس دانشکده‌های فنی
دانشکده برق و کامپیوتر



درس تحلیل ها و سیستم های داده های حجیم

تمرین شماره سه

نام و نام خانوادگی :

سحر رجبی

شماره دانشجویی :

۸۱۰۱۹۹۱۶۵

اردیبهشت ۱۴۰۱

فهرست گزارش سوالات (لطفاً پس از تکمیل گزارش، این فهرست را بهروز کنید.)

3.....	پیش‌پردازش
4.....	سوال ۱
10.....	سوال ۲
16.....	سوال ۳
19.....	سوال ۴
28.....	سوال ۵

پیش پردازش

برای استخراج داده‌های خواسته شده، از کدهای تمرین قبلی استفاده شده است که داده‌ها را در یک csv ذخیره می‌کند. برای اضافه کردن فیلدهای خواسته شده هم مشابه قبل با استفاده از inspect-

بررسی‌ها را انجام دادیم و اطلاعات مورد نیاز را با کمک BeautifulSoup استخراج کردیم.

در نهایت با استفاده از csv.DictReader هر خط از csv را به صورت یک dictionary خواندیم تا

نهایتاً داده‌ها را در یک فایل json ذخیره کنیم تا هر زمانی که یک فیلد در یک داده وجود نداشت، فیلد متناظر آن در نمونه‌ی داده هم وجود نداشته باشد. همچنین tags را به صورت لیست و تاریخ را با فرمت yyyy-mm-ddTh:i ذخیره کردیم.

کدهای پیاده‌سازی شده در فایل‌های irna_parser.py و isna_parser.py وجود دارد و در نهایت در collect_data.py استفاده شده‌اند که هر سه‌ی این کدها در فولد pre-process وجود دارد.

سوال ۱ -

با استفاده از docker-compose.yml داده شده، کانتینرهای مورد نیاز بدون مشکل ایجاد شدند اما در این کانفیگ، برای presto تنها یک node داشتیم که هم نقش coordinator و هم نقش worker را ایفا می کرد. برای اضافه کردن یک worker دیگر به presto باید کارهای زیر را انجام می دادیم.

اول اینکه برای بالا آوردن یک کانتینر presto از Dockerfile ای که در داخل فolder presto/presto وجود دارد استفاده شده است. هر node در presto به تعدادی config احتیاج دارد که شامل فایل های node.properties، log.properties، jvm.config، config.properties است. فایلی که بین نودهای node.properties و config.properties worker و coordinator متفاوت است، فایل های config.properties و node.properties است. برای config.properties documentation در سه کانفیگ مختلف برای نودهای coordinator و worker و نودی که نقش هر دو را ایفا کند ارائه شده است. برای اضافه کردن یک worker ما در کنار فolder etc-coordinator یک فolder etc-worker اضافه می کنیم و فایل های فolder اول را در آن قرار می دهیم اما در فایل config.properties کانفیگ ارائه شده در سایت presto که مطابق زیر است را قرار می دهیم.

```
coordinator=false
http-server.http.port=8080
query.max-memory=50GB
query.max-memory-per-node=1GB
discovery.uri=http://presto-coordinator:8080
```

همچنین در فایل node.id presto-worker هم مقدار node.properties با قرار می دهیم.

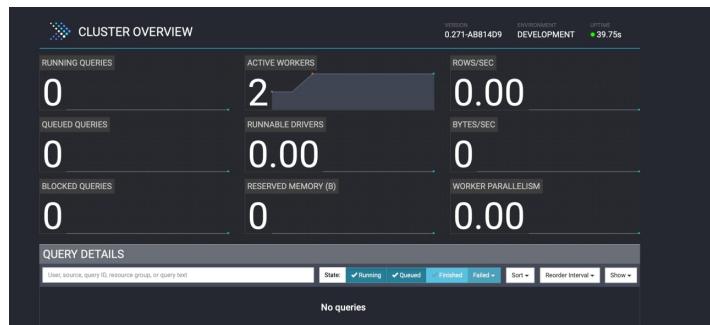
دامرایی که در کنار این فolders وجود دارد، با گرفتن PRESTO_NODETYPE مشخص می کند که از روی کدام یک از دو فolder etc-worker و يا etc-coordinator باید کانتینر را build کند.

در نهایت هم در فایل docker-compose.yml دیگر برای نod worker به صورت زیر اضافه می کنیم.

```
presto-worker:
  image: prestodb-worker:0.271
  build:
    context: ./presto
    dockerfile: Dockerfile
    args:
      PRESTO_VERSION: 0.271
      PRESTO_NODETYPE: "worker"
    container_name: presto-worker
    hostname: presto-worker
    environment:
      SERVICE_PRECONDITION: "presto-coordinator:8080"
  volumes:
    - presto_worker:/var/presto/data
    - ./presto/etc-worker:/opt/presto/etc
```

بعد از اضافه کردن این نود، باید باز دیگر docker-compose up را اجرا کنیم تا نود worker هم اضافه شود.

تصویر زیر بعد از اضافه کردن نود worker از دشبورد presto ثبت شده است.



بخش اول)

(الف)

برای اتصال presto به hive باید در پوشه‌ی /etc/catalog در هر کانتینر، یک hive.properties داشته باشیم (پوشه‌های etc-worker و etc-coordinator که در local ما هستند با استفاده از volumes در کانتینر مپ شده‌اند پس ما تغییرات را در این پوشه‌ها اعمال می‌کنیم). فایل catalog را در پوشه‌ی metastore ایجاد می‌کنیم که دو خط زیر در آن قرار دارد:

```
connector.name=hive-hadoop2
```

```
hive.metastore.uri=thrift://hive-metastore:9083
```

در اینجا uri را برابر با localhost:9083 قرار دادیم، چرا که اگر برابر با metastore 9083 در کانتینر localhost به کانتینر نod ما اشاره می‌کند. ما می‌توانیم که پورت 9083 در metastore را به پورت 9083 در لوكال وصل کنیم، اما اين کافيگ در يك نود presto قرار دارد که هيچ يك از اين دو نیست، در نتيجه باید از نام metastore استفاده کنیم تا در نتورک ساخته شده در نودهای dockercompose آن را بشناسد. در ضمن اين فایل را در هر دو نود coordinator و worker قرار دادیم.

(ب)

مطابق توضیحات تمرين قبل، برای ایجاد يك پوشه در HDFS از دستور

```
hdfs dfs -mkdir -p /data/news/
```

استفاده کردیم که -mkdir دستور ساخت پوشه‌ی جدید را می‌دهد و -p هم فلگی است که اگر پوشه‌های parent آن وجود نداشتند، آن‌ها را هم بسازد.

برای کپی کردن داده‌ها از حافظه‌ی محلی به hdfs هم، با دستور

```
hdfs dfs -put <local files path> <path in hdfs>
```

فایل‌ها را از حافظه‌ی محلی در hdfs آپلود می‌کنیم.

در نهایت برای ست کردن replication factor برابر ۳، از دستور بعدی استفاده کردیم.

```
hdfs dfs -setrep -R 3 <data path in hdfs>
```

با استفاده از -R، replication factor تمامی فایل‌های داخل دیرکتوری را تغییر می‌دهیم.

برای دانلود داده از سایت imdb و قرار دادن آن در hdfs هم از دستور زیر استفاده کردیم.

```
curl <download-link> | gunzip -c | hdfs dfs -put - /data/file-name
```

برای ساخت جداول هم از کدهای تمرین قبلی استفاده کردیم که فایل آن‌ها در کنار تمرین آپلود شده است. برای اجرای این کوئری‌ها، درdbeaver مستقیماً به hive وصل شدیم و جداول را در خود hive ایجاد کردیم.

(ج)

برای اجرای کوئری‌ها در بستر presto، coordinator را به نود presto متصل کردیم و کوئری‌ها را برای آن ارسال کردیم. کوئری‌ها مطابق قبل هستند اما برای اشاره به جداول، که در قبل از استفاده می‌کردیم، این بار باید از نام catalog یا در متن خود کوئری‌ها نام جداول را به صورت use <database_name> اقدام کنیم یا در متن خود use <catalog_name.database_name> catalog_name.database_name.table_name بیان کنیم. فایل کوئری‌ها هم آپلود شده است.

(د)

	Wall-time in Hive	Execution-time in Hive	Wall-time in Presto	Execution-time in Presto	Peak memory in Presto
news_q1	8s	4s	8s	6.99s	0
news_q2	3s	3s	1.37s	1.15s	0
news_q3	38s	38s	2.62s	2.39s	0
news_q1_partitioned	2s	1s	1.35s	1.21s	0
news_q2_partitioned	3s	3s	2.05s	1.91s	0
news_q3_partitioned	28s	28s	1.14s	1.01s	0
imdb_q1	62s	61s	13.3s	13s	2.65MB
imdb_q2	180s	179s	21.4s	21.2s	597MB

تقریباً در تمامی کوئری‌ها، سرعت اجرای presto بیشتر از hive بوده است. در کوئری‌هایی مثل imdb_q2 یا news_q3 این تفاوت بسیار قابل توجه است. به صورت کلی presto می‌تواند performance را تا حد بسیار خوبی افزایش بدهد.

بخش دوم)

(الف)

برای اضافه کردن کانتینر mongoDB و mongo-express به کانتینرهای قبلی، بخش زیر در docker-compose اضافه شده است.

```
mongo:
  image: mongo
  restart: always
  container_name: mongo
  hostname: mongo
  volumes:
    - mongoDB:/data/db
    - /data:/project/data/

mongo-express:
  image: mongo-express
  restart: always
  ports:
    - 8081:8081
  hostname: mongo-express
  container_name: mongo-express
  environment:
    SERVICE_PRECONDITION: "mongo"
    ME_CONFIG_MONGODB_URL: mongodb://mongo:27017/
```

(ب)

برای اضافه کردن داده‌های استخراج شده به MongoDB هم از mongoimport استفاده کردہ‌ایم که با دستور زیر داده‌ها را در یک collection قرار می‌دهد.

```
mongoimport -d newsdb -c news --type JSON --file "news.json" --jsonArray
```

(ج)

مانند توضیحات ارائه شده برای اضافه کردن کاتالوگ hive به presto در قسمت‌های قبل، در پوشه‌ی catalog در هر دو پوشه‌ی etc-worker و etc-coordinator یک فایل با نام mongodb.properties ایجاد می‌کنیم که در آن دو خط زیر قرار دارد.

```
connector.name=mongodb
```

```
mongodb.seeds=mongo:27017
```

(د)

فایل کوئری‌های نوشته شده در کنار گزارش آپلود شده‌اند. اما در ادامه توضیح کوتاهی برای چگونگی پیاده‌سازی هر یک از کوئری‌ها و خلاصه‌ای از نتیجه‌ی آن‌ها قرار می‌دهیم.
کوئری اول:

برای به‌دست آوردن برچسب‌های یکتای استفاده شده در دسته‌های «سرвис اجتماعی» و «جامعه» از دو تابع presto کمک گرفتیم. ابتدا با استفاده از array_agg تمامی برچسب‌ها که خود در یک لیست قرار داشتند را در یک لیست دیگر ریختیم. سپس با تابع flatten لیستی از لیست‌ها را به تنها یک لیست

تبديل کردیم و در واقع اعضای لیست‌های داخلی را در لیست بزرگ‌تر قرار دادیم. سپس با استفاده از `distinct` برچسب‌های تکراری را حذف کردیم. نتیجه به صورت زیر است:

Name	Value
۰	اختصاصی ایران
۱	بهداشت و سلامت
۲	وینامن
۳	وبروز کرونا
۴	بازنشستگان
۵	حق و حقوق
۶	صندوق بازنشستگی کشوری
۷	شهر تهران
۸	محسن منصوری
۹	ت ریاست جمهوری در امور انان و خارج از
۱۰	جلسه شورای اداری
۱۱	سازمان تامین اجتماعی
۱۲	میرهاشم موسوی
۱۳	اشغال زاید
۱۴	بهبود فضای کسب و کار
۱۵	سازمان نظام روزنامه‌سازی و مشاوره
۱۶	تزوییست
۱۷	خوزستان
۱۸	نهی نیروی انتظامی جمهوری اسلامی ایران
۱۹	سازمان مخاطط محیط زیست
۲۰	مدیریت پسماند
۲۱	وزارت کشور
۲۲	انجمن

کوئری دوم:

برای پیدا کردن خبرهای مرتبط به کرونا با توجه به کلمات استفاده شده در برخی فیلدهای خبرها، از تابع `regexp_like` برای فیلدهای `title`, `text` و `summary` استفاده کردیم. به این صورت که در این فیلدها به دنبال کلماتی که با پترن `regex` ما- که با `or` کلمات داده شده است- مطابقت دارد می‌گردیم. برای `tag`ها هم از تابع `any_match` استفاده می‌کنیم که در آن برای هر `tag` شرط `regexp_like` قبلی را که اشاره کردیم اجرا می‌کنیم.

کوئری سوم:

برای این کوئری از `nested_query` استفاده کردیم.

ابتدا فرکانس تکرار هر برچسب را با استفاده از تابع `array_frequency` که بر روی `flatten(array_agg(tags))` صدا شده است به دست می‌آوریم. برای این کار، داده را بر روی ترکیب `main_category` و `sub_category` در نتیجه `array_agg` تمام تگ‌های یک گروه را در یک لیست جمع می‌کند. `Flatten` هم لیست لیست‌ها را تبدیل به یک لیست یک‌بعدی می‌کند و `add_frequency` هم فرکانس تکرار هر برچسب را بدست می‌آورد. خروجی `map` یک `add_frequency` است که با استفاده از `unnest` آن را به دو فیلد `tag` و `freq` می‌شکانیم. و با استفاده از تابع `rank` با مرتب کردن روی `freq` رنک را مشخص می‌کنیم و نهایتاً با فیلتر رنک ۱ تا ۵ خروجی را مشخص می‌کنیم.

نتیجه به صورت زیر است:

main_category	sub_category	_col2
اقتصاد	ارتباطات	{همراوه اول=1, ایرانسل=1, ایران=1, روابط عمومی ایرانسل=1, اکهی بازرگانی=1, تبلیغات آنلاین=1}
فرهنگ	کتاب و ادبیات	ت متحده آمریکا=1, ادبیات=1, نمایشگاه بین المللی کتاب تهران=1, نقد کتاب=1, علی رمضانی=1, اکهی بازرگانی=1, کتابخوانی=1}
استان‌ها	قزوین	کمینه امداد امام خمینی=2, نوروز=2, استان قزوین=12, شهرداری قزوین=2, سازمان راهداری و حمل و نقل جاده ای=2}
استان‌ها	خوزستان	دن و تجارت=6, ابادان=5, سیدابراهیم ریس رییس جمهوری=5, اهواز=16, دزفول=6, خوزستان=34, شهید رضوی=2, سازمان راهداری و حمل و نقل جاده ای=2, نفت مسجد سلیمان=5
فرهنگ	فرهنگ و ارشاد اسلامی	د امامی=1, شوای فرهنگ عمومی=1, فرهنگ=1, تیپرسکان=1, پروار=1, تنگه هرمز=1, سازمان هواشناسی=1, ایرنا=1, سروایا=1, ستد مل=1, دریا عم مان بنادر و دریانوردی=1, تیپرسکان=1, پروار=1, تنگه هرمز=1, سه=1, جنت الله عبد الملک=1, بار طلاق=1, اتیلهات آنلاین=1, بار طلاق=1, بیرونی=1, بیمه=1, باک و بیمه=1, پژوهش‌های دانشگاهی=1, سمندان=1, مازندران=1, زلزله=1, بهبود فضای سبب و کار=1, علم و اموزش=1, دانشگاه شهید بهشتی=1, دانشگاه هنر اسلامی=1, دومنی نمایشگاه محایی کتاب تهران=1, داشجو=1, فرانسه=1, دانشگاه هنر اسلامی=1, کتاب دیجیتال=1, کنکری=1, دومنی نمایشگاه محایی کتاب تهران=1, ایرنا=1, آفاق ایران=1, چین=1, تجارت=1, ایرنا=1, آسیای جنوبی=1, سازمان تلفن‌افسان اسلامی=1, ایران=1, هرمزگان=3, مهدی دوست=3, بندعباس=6, بستک=3, سیاه پاسداران انقلاب اسلامی=4, قرچک=7, پاکدشت=6, ورامین=15, شهرستان ری=5, ایرنا=4}
اقتصاد	باک و بیمه	ت=1, اتحادیه طلا و جواهر تهران=1, مدیرعامل بعثه سرمد=1, سه=1, بیرونی=1, بیمه=1, بازار طلاق=1, اتیلهات آنلاین=1, بار طلاق=1, بیمه=1, باک و بیمه=1, پژوهش‌های دانشگاهی=1, سمندان=1, مازندران=1, زلزله=1, بهبود فضای سبب و کار=1, علم و اموزش=1, دانشگاه شهید بهشتی=1, دانشگاه هنر اسلامی=1, دومنی نمایشگاه محایی کتاب تهران=1, داشجو=1, فرانسه=1, دانشگاه هنر اسلامی=1, کتاب دیجیتال=1, کنکری=1, دومنی نمایشگاه محایی کتاب تهران=1, ایرنا=1, آفاق ایران=1, چین=1, تجارت=1, ایرنا=1, آسیای جنوبی=1, سازمان تلفن‌افسان اسلامی=1, ایران=1, هرمزگان=3, مهدی دوست=3, بندعباس=6, بستک=3, سیاه پاسداران انقلاب اسلامی=4, قرچک=7, پاکدشت=6, ورامین=15, شهرستان ری=5, ایرنا=4}
علم و اموزش	دانشگاهیان	{آفاق ایران=1, چین=1, تجارت=1, ایرنا=1, آسیای جنوبی=1, سازمان تلفن‌افسان اسلامی=1, ایران=1, هرمزگان=3, مهدی دوست=3, بندعباس=6, بستک=3, سیاه پاسداران انقلاب اسلامی=4, قرچک=7, پاکدشت=6, ورامین=15, شهرستان ری=5, ایرنا=4}
علم و اموزش	بازارگان	
اقتصاد	جهان	
استان‌ها	استان‌ها	
استان‌ها	تهران	

کوئری چهارم:

برای این کوئری، ابتدا با تابع `regexp_extract` زمان را استخراج کردیم (همچنین از `regexp_replace` هم برای تمیز کردن نتیجه استفاده کردیم) و نهایتاً عدد ساعت را به `int` کست کردیم و آن را تقسیم بر ۶ کردیم. نام این نتیجه را `day_time` گذاشتیم و نتیجه را بر اساس آن `group by` کردیم و `count` هر دسته را گزارش کنیم.

نتیجه به صورت زیر است:

locked	123 _col0
1	312
2	908
3	1,790
4	982

(۵)

زمان انجام کوئری در جدول زیر گزارش شده است.

	Wall-time	Execution-time	memory in Presto
Q1	254ms	250ms	0
Q2	1.92s	1.89s	2.66
Q3	4.54s	4.52s	3.33
Q4	281ms	264ms	0

مقایسه با قسمت دوم به شرح زیر است:

به غیر از کوئری سوم، که ملزم به استفاده‌ی زیاد از تابع‌های مخصوص به `presto` مثل `flatten` یا `arr_agg` بودیم، زمان اجرای بقیه‌ی کوئری‌ها در `presto` کمتر است.

سوال ۲-

(الف)

- تمامی برچسب‌های یکتا در در دسته‌ی خبری اجتماعی یا جامعه.

برای انجام این کوئری از پایپ‌لاین در aggregation مونگو استفاده کردیم. به این صورت که ابتدا داده را با استفاده از \$match طوری فیلتر می‌کنیم که دو دسته‌ی دلخواه باقی بمانند، سپس با استفاده از \$unwind فیلد tags را از حالت لیست خارج می‌کنیم و در مرحله‌ی بعد با استفاده از یک \$group که id_ را در آن برابر با null قرار می‌دهیم، از \$addToSet استفاده می‌کنیم و تمامی tag‌ها را داخل یک ست می‌ریزیم و به این ترتیب تگ‌های یکتای استفاده شده در دسته‌ی اجتماعی و جامعه را خواهیم داشت. بخشی از خروجی به شکل زیر است:

```
tags_s: [
  'اطفالی حیرق',
  'اختصاصی ایران',
  'بارش باران و برف',
  'اسازمان حفاظت محیط زیست',
  'جمهوری اسلامی ایران',
  'وزارت آموزش و پرورش',
  'کاد استر',
  'سازمان نظام روانشناختی و مشاوره',
  'ومیکرون',
  'کمیته امداد امام خمینی',
  'ادارو',
  'وزارت کشور',
  'سفر نوروزی ۱۴۰۱',
  'تصادفات چاده ای',
  'پروتکلهای بده اشتی',
  'اسدوار سید تمیم حسینی',
  'استاد کرونا',
  'روانشناسی',
  'هوای ناسالم برای گروه های حساس',
  'روابط خاتون ادکن',
  'وزیر کشور',
  'امداد هوایی']
```

- تعداد خبرهای با کلمات مربوط به کروناویروس در فیلدهای مختلف.

برای این کوئری از قابلیت regex استفاده کردیم. به این صورت که در روی فیلدهای title، summary و text به دنبال زیرشته‌هایی با فرمت regex حاصل از or کلمات داده شده گشته‌یم و برای tag هم از in برای لیستی از pattern استفاده کردیم و تمامی شرط‌ها را با یکدیگر or کردیم. از این قسمت در بخش aggregate \$match در \$match استفاده کردیم و نهایتاً تعداد این خبرها را برابر با ۳۰۲ خبر بدست آوردیم.

- افزودن تعداد پاراگراف‌ها.

برای افزودن تعداد پاراگراف‌ها، ابتدا رکوردهایی که فیلد text دارند را در aggregation انتخاب کردیم و سپس با استفاده از

```
{$size: {$split: ["$text", "#"]}}
```

و با توجه به این نکته که در پالایش داده، پاراگراف‌ها را توسط # از یکدیگر جدا کرده‌ایم، تعداد پاراگراف‌ها را در هر خبر بدست می‌آوریم. (#ها در متن اصلی قبل از مشخص کردن پاراگراف‌ها حذف شده‌اند). در نهایت فیلدی به صورت زیر به داده‌ها اضافه شده است:

```
    "حمله روسیه به اوکراین",
],
'main-category': '#سرویس ویدئو',
'sub-category': 'خبر',
'short-link': 'https://www.isna.ir/news/140012262',
'reporter-code': '71602',
'week-day': 'پنجشنبه',
'editors': 'حسین هرمزی',
paragraph: { count: 14 },
},
{
_id: ObjectId("6287ca72412dc4963efea731"),
'reporting-service': 'isna',
code: '1400122721393',
```

- افزودن تاریخ میلادی.

برای تبدیل تاریخ شمسی به میلادی، با استفاده از dateAdd تاریخ را دریافت می‌کنیم و فرمت پارس آن را هم به آن می‌دهیم؛ (نکته: به علت متغیر بودن تعداد روز در ماه‌های مختلف، عدد ۳۰ در قسمت ماه را با ۲۸ جایگزین می‌کنیم و این باعث بدست آوردن تاریخ تقریبی (با دو روز تأخیر) خواهد بود) سپس تعداد تفاضل دو سال شمسی و میلادی را به آن می‌دهیم تا به تاریخ میلادی برسیم.

- اطلاعات ۲۰ خبر با بیشترین تعداد پاراگراف.

در این بخش تنها با استفاده از تابع find، در حالی که با استفاده از sort بر روی فیلد paragraph.count به صورت نزولی سورت می‌کردیم، مشخص کردیم که چه فیلد‌هایی برگردانده شود و با است کردن محدودیت ۲۰ رکورد، ۲۰ خبر مد نظر را گرفتیم. بخشی از نتیجه به این صورت است:

```
{
  "این‌بار مرگ را نوشتنند",
  date: '1400-12-26T00:12',
  reporters: 'فدا ولیپور',
  paragraph: { count: 221 }
},
{
  "اما قادر هسته‌ای هستیم و از هیچکس نمی‌ترسیم",
  date: '1400-12-25T11:43',
  paragraph: { count: 158 }
},
{
  "چه کسانی نامزدهای نهایی برترین‌های قدر ورزش را انتخاب کردند؟",
  date: '1400-12-27T12:04',
  reporters: 'سید مهدی خادم‌نیا',
  paragraph: { count: 126 }
},
{
  "رازی که "شیخ" را از سایر دانشگاه‌ها مقنایز می‌کند",
  date: '1400-12-21T04:22',
  reporters: 'قاطمه حسین‌پور'
```

(ب)

- خبرهای مربوط به دانشگاه در بازه‌ی زمانی مشخص.

برای این کوئری، در دستور aggregate، در استیج match با استفاده از regex فیلد text را طوری فیلتر کردیم که اقلاً یکی از ۳ کلمه‌ی اشاره شده را داشته باشد (مشابه کوئری بخش قبلی) و بعد مجدداً با استفاده از regex یک روز را برای فیلد date مشخص کردیم که با پtern '1400-12-25'^ مشخص شد و در استیج project عنوان خبر را برگرداندیم، بخشی از خروجی به صورت زیر است:

```
{
  title: { }, 'نتویم 2500 و اندی پاره شد' },
  title: { }, 'پیشنهاد وزارت علوم برای توسعه فناوری هوش مصنوعی در کشور' },
  title: { }, 'اظطر تله جمعیتی برای ایران' },
  title: { }, 'برگزاری نشست "مقاله: مشارکت در پیشبرد علم با ابزار ارتقا"' },
  title: { }, 'حقوق نمایندگان مجلس، اعضا شورای نکهیان و مجمع تشخیص مفاسد می‌شود' },
  title: { }, 'آشنایی با وکیل زمین' },
  title: { }, 'تعیین تکلیف نحوه آموزش دانشجویان دانشکاه تهران در سال ۱۴۰۱' },
  title: { }, 'آغاز همکاری مجلس و دانشگاه علمی کاربردی در حوزه مهارت آفرینی' },
  title: { }, 'مهارت آموزی به فارغ التحصیلان دانشگاهی' },
  title: { }, 'استنادهای متعدد نسخه مربیتی را جمع آوری کرد' },
  title: { }, 'نتایج دو آزمون دانشگاه آزاد اعلام شد' },
  title: { }, 'حساب‌شدن کار مرکز الکو اسلامی ایران پیشرفت پس از ابلاغ سند'
```

- ۵ برچسب پر تکرار در هر زیردستهی خبری.

برای این کوئری، خبرهایی که دسته‌ی اصلی و زیردسته دارند را جدا می‌کنیم (مراحل، استیج‌های هستند) همچنین با استفاده از unwind تگ را از حالت لیست خارج می‌کنیم. سپس داده را بر روی زوج دسته‌ی اصلی و زیر دسته و مقدار tag گروه می‌کنیم، و در هر گروه، تعداد را با \$sum: 1 می‌شماریم. سپس دیتا را بر اساس تعداد سورت می‌کنیم، بار دیگر دیتا را گروه می‌کنیم، اما این بار تنها روی زوج دسته‌ی اصلی و زیر دسته، و tagها که در حال حاضر سورت هستند را در یک لیست پوش می‌کنیم. در قسمت projection با استفاده از slice، ۵ آرایه‌ی اول از آن لیست را نگه می‌داریم. نتیجه در زیر قابل مشاهده است:

```
{
  _id: { m: 'اسپاپ', s: 'احزاب و تشکلها' },
  tags: [
    'انگلستان',
    'امصار جمهوری تهران',
    'امصار معظم رهبری',
    'اسید محمدحسن ایوبی قردن',
    'شهید امدادی حرم'
  ],
  {
    _id: { m: 'ایزد', s: 'استان‌ها' },
    tags: [ 'ایزد', 'امهریز', 'ایزکوه', 'امینی', 'اصنایع دستی' ]
  },
  {
    _id: { m: 'اصفهان', s: 'استان‌ها' },
    tags: [
      'اصفهان',
      'ازابنده رود',
      'قویتبال',
      'ارودخانه‌های کوهستانی',
      'استاد تنظیم بازار'
    ]
  },
  {
    _id: { m: 'آذربایجان', s: 'سينما و تئاتر' }
  }
}
```

- تعداد کلمات هر خبرنگار در هر ماه و ۱۰ خبرنگار پر حرف.

در aggregate ابتدا با match خبرهایی که هم خبرنگار، هم متن و هم تاریخ دارند را جدا کردیم. سپس در project با استفاده از دستور split متن را بر اساس فاصله‌ها جدا کردیم و با size تعداد کلمات را بدست آوردیم. سپس ماه را هم با استفاده از arrayElemAt که بر روی split تاریخ انجام شد به دست آوردیم. در قسمت group داده را بر روی ماه و نام خبرنگار group کردیم و مجموع جملات برای هر خبرنگار در هر ماه را با sum به دست آوردیم. در نهایت ابتدا با ماه، و سپس با تعداد کلمات در هر ماه سورت کردیم و نتیجه‌ای به شکل زیر بدست آوردیم:

```
{ _id: { month: '10', reporter: 'مسنی وفایی' }, total_word: 1912 },
{ _id: { month: '10', reporter: 'زهرا عطایی‌پور' }, total_word: 1822 },
{
  _id: { month: '11', reporter: 'کبیرا حسینزاده' },
  total_word: 5316
},
{
  _id: { month: '11', reporter: 'اصفهونه سوہانی' },
  total_word: 3840
},
{
  _id: { month: '11', reporter: 'زهرا حیدری' }, total_word: 3437 },
{ _id: { month: '11', reporter: 'مسنی وفایی' }, total_word: 3142 },
{ _id: { month: '11', reporter: 'امنه کیمیان' }, total_word: 1987 },
{ _id: { month: '11', reporter: 'امیر حسروفرز' }, total_word: 1626 },
{ _id: { month: '11', reporter: 'زهرا طایپور' }, total_word: 1483 },
{ _id: { month: '11', reporter: 'سیده حسنلو' }, total_word: 1028 },
{ _id: { month: '11', reporter: 'ایمانه قاسمی' }, total_word: 751 },
{ _id: { month: '11', reporter: 'سیده حسنلو' }, total_word: 728 },
{ _id: { month: '11', reporter: 'امنه شفیعی' }, total_word: 728 },
{ _id: { month: '11', reporter: 'پرسنل توکلی' }, total_word: 403 },
{ _id: { month: '11', reporter: 'فائزه طاهری' }, total_word: 295 },
{
  _id: { month: '11', reporter: 'امنیت‌ساده سید رضا' },
  total_word: 262
},
{
  _id: { month: '11', reporter: 'الهیلا دادمهر (اکیر)' },
  total_word: 102
},
{ _id: { month: '12', reporter: 'اصفهونه حسنلو' }, total_word: 11997 },
{
  _id: { month: '12', reporter: 'کبیرا حسینزاده' },
  total_word: 9720
},
```

حالا برای اینکه بتوانیم ۱۰ خبرنگار را بگیریم، یک بار دیگر بر روی ماه group می‌کنیم و نام خبرنگاران را در یک لیست push می‌کنیم. سپس با استفاده از slice، ۱۰ نام اول در آن لیست را در project نگه می‌داریم و در خروجی می‌نویسیم.

```
{ _id: { month: '4' }, reporters: [ 'فائزه طاهری' ],
{ _id: { month: '3' }, reporters: [ 'الهیلا دادمهر (اکیر)' ],
{ _id: { month: '6' }, reporters: [ 'حسین شایسته', 'فائزه طاهری', 'اصفهونه حسنلو' ],
{
  _id: { month: '7' },
  reporters: [ 'الهیلا دادمهر (اکیر)', 'فائزه طاهری', 'اصفهونه حسنلو' ]
},
{ _id: { month: '2' }, reporters: [ 'الهیلا دادمهر (اکیر)' ],
{ _id: { month: '10' }, reporters: [ 'مسنی وفایی', 'زهرا عطایی‌پور' ],
{ _id: { month: '8' }, reporters: [ 'اصفهونه حسنلو' ],
{
  _id: { month: '12' },
  reporters: [
    'اصفهونه حسنلو',
    'کبیرا حسینزاده',
    'اصفهونه سوہانی',
    'امنه کیمیان',
    'امیر حسروفرز',
    'امیر دژایی',
    'اعظم سندکلار',
    'امادی قره‌نگ زاده',
    'امنه حسینی‌پور',
    'محمدحسینی رضوانی‌فرد',
    'سید مهدی خادمی'
  ]
},
{
  _id: { month: '9' },
  reporters: [
    'زهرا عطایی‌پور',
    'پرسنل توکلی',
    'اصفهونه ایمانیان',
    'فیبا عبد‌الله‌خانی'
  ]
}
}
```

- حذف کد خبرنگارهای بدون نام.

برای این کار ابتدا رکوردهایی که نام ادیتور داشتند و نام خبرنگار نداشتند را پیدا کردیم و با استفاده از unset فیلد reporter-code را در آنها حذف کردیم.

- تعداد خبرها در بازه‌های ۶ ساعته‌ی روز.

برای این کار، در ساختار aggregate ابتدا رکوردهایی که فیلد تاریخ دارند را جدا کردیم، سپس با ساختار project، قسمت زمان را از بخش تاریخ جدا کردیم و در فیلدی موقت ذخیره کردیم. سپس در یک project دیگر، قسمت ساعت را از زمان جدا کردیم، آن را به int کست کردیم، با استفاده از divide عدد ساعت را بر ۶ تقسیم کردیم و مجدد به int کست کردیم. در نهایت بر روی hour داده را group کردیم و count را حساب کردیم. (برای قسمتهای جدا کردن، از ترکیب arrayElemAt و مشابه قسمتهای قبل استفاده کردیم). نتیجه به صورت زیر شد و عدد ۰ معادل ۶ ساعت اول، ۱، ۶ ساعت دوم و غیر می‌باشد.

```
{ _id: 2, count: 895 },|  
{ _id: 0, count: 156 },  
{ _id: 3, count: 454 },  
{ _id: 1, count: 491 }
```

- انتقال خبرهای دسته‌ی ورزشی.

برای این کار، با ساختار aggregate ابتدا خبرهایی که در دسته‌ی اصلی خود کلمه‌ی ورزش را داشتند با (از ورزش برای پوشش هر دو خبرگزاری استفاده شده) فیلتر می‌کنیم و سپس با استفاده از \$out نتیجه را در یک کالکشن دیگر می‌نویسیم. سپس با استفاده‌ها از deleteMany مشابه قبل خبرها را فیلتر کرده و از دیتابیس حذف می‌کنیم.

جدول زیر، زمان‌های اجرای کوئری‌های مشترک با سوال قبل است (عنوان سوال‌ها هم مطابق با بخش قبل است و مقایسه در سوال قبلی انجام شده است).

	Execution-time	memory in Presto
Q1	1.283s	0
Q2	1.373s	2.66
Q3	1.321s	3.33
Q4	1.447ss	0

(ج)

در این بخش زمان سیستم گزارش شده است. در ضمن بین اجرای کوئری‌ها فاصله است تا از اثر cache در امان باشیم.

- کوئری ۱ بخش الف.

زمان اجرا قبل از ایندکس ۱۴۵S بود. یک ایندکس بر روی main-category ایجاد می‌کنیم. زمان اجرا این بار برابر ۱۳۸S است. که به وضوح کاهش یافته است.

- کوئری ۵ بخش الف.

زمان اجرا قبل از ایندکس ۱۴۶S بود. این بار بر روی paragraph.count ایندکس ایجاد کردیم. نتیجه برابر با ۱۶۹S است که تفاوت بسیار کمی با یکدیگر دارند.

- کوئری ۲ بخش ب.

زمان اجرا قبل از ایندکس ۱۵۲S بود. این بار علاوه بر اینکه ایندکس main_category داریم، یک ایندکس هم بر روی sub_category ایجاد کردیم. این بار زمان برابر ۱۴۶S است که تفاوت زیادی ندارد.

- کوئری ۳ بخش ب.

زمان اجرا قبل از ایندکس ۱۵۷S بود. یک ایندکس بر روی reporters ایجاد کردیم و زمان به ۱۴۵S رسید.

- کوئری ۵ بخش ب.

زمان اجرا قبل از ایندکس ۱۶۴S بود. بعد از ایجاد یک ایندکس بر روی تاریخ، برابر با ۱۴۳S شد.

به صورت کلی، ایجاد ایندکس می‌تواند زمان ما را کاهش بدهد اما در اضافه کردن آن، باید بسیار توجه کرد. برای مثال، در کوئری ۵ بخش ب، اضافه کردن ایندکس بر روی تاریخ تاثیر بهسزایی نمی‌توانست در دیتای ما داشته باشد. همچنین به علت کم بودن داده‌ها در این تمرین، تفاوت عملکرد ایندکس به خوبی مشخص نیست و از طرفی با ایجاد ایندکس، ما فضای ذخیره‌سازی را از دست می‌دهیم و در انتخاب آن باید به دقیقیت عمل کنیم.

سوال ۳-

برای نصب برنامه، به سادگی با دستور `brew install redis` اقدام به نصب آن می‌کنیم و بعد از اجرای `redis-cli` با `redis-server` می‌توانیم با `redis` به آن وصل شویم.

در ادامه ساختار استفاده شده برای رسیدن به پاسخ کوئری‌های خواسته شده را توضیح خواهیم داد.

برای نوشتن داده‌ها در `redis` یک بار فایل `data.json` را پیمایش کردیم و داده‌ساختارهای مورد نیاز را تشکیل دادیم.

برای کوئری اول از دو `set` و یک `hash` استفاده کردیم. به این صورت که خبرهایی که در دسته‌ی ورزشی قرار داشتند را مشخص کرده، و نام `reporter` را در `set`‌ای با نام `reporter` اضافه می‌کنیم و کد خبرنگار را هم در `set`‌ای با نام `reporter-codes` تا در نهایت بتوانیم لیست نام خبرنگاران و کد خبرنگارانی که در دسته‌ی ورزشی خبر منتشر کرده‌اند را با تابع `smembers` دریافت کنیم. همچنین از آنجایی که برخی خبرها، تنها نام خبرنگار را داشتند، برخی تنها کد خبرنگار را داشتند و برخی هم هر دو، از یک `hash` با نام `reporter-code-mapping` استفاده کردیم که در آن `key` نام خبرنگار، و `value` کد خبرنگار است و آن را با استفاده از صفحاتی که در آن‌ها هم نام خبرنگار و هم کد خبرنگار وجود دارد پر می‌کنیم. با این روش می‌توانیم اشتراک دو مجموعه‌ی `reporter` و `reporter-codes` را بدست بیاوریم و زمانی که می‌خواهیم تعداد خبرنگاران فعال در دسته‌ی ورزشی را گزارش کنیم، طبق قانون مجموعه‌ها، طول `reporter` و `reporter-codes` را با هم جمع می‌کنیم و تعداد اعضای `reporter-codes-mapping` را از مجموع آن‌ها کم می‌کنیم.

نتیجه‌ی کوئری اول به صورت زیر است:

```
reporters-list:
[مهاری دودخواهی
بعضیون عین الدجوازی -بعضیون عین الدجوازی
پیروز احمدی
شروعی دیپلماتی
بعضیون عین الدجوازی
پیغمباریان شفیقی
روزبه رادمهر
طیبیله سانکهور
پریختی پیغمبری
امیدعلیی عوفی خوار
پریسا نسلیه میلان
بعضیون شهیدواری
پیغمبار کردیمی
وحید دینی
آقایی خلیلی
مهدی چیلو
بعضیون پیغمبران
اصغر عین الدلیلی
reporter-codes:
[71649, '71552', '50228', '71572', '71575', '71613', '71518', '71617', '90847', '98111', '71576', '71506', '71468', '71468', '50213', '71536', '50084']
num of reporters:
17
```

برای کوئری دوم از یک `sorted set` با نام `social-tagS` استفاده کردیم که با مشاهده‌ی هر `tag` در دسته‌ی اجتماعی، با دستور `zincrby` یکی به تعداد آن `tag` اضافه می‌کند و از آنجایی که یک `sorted set` است، در نهایت این مقادیر به صورت `sort` شده قرار دارند. در نهایت برای زدن کوئری از دستور `zrange` استفاده می‌کنیم که رنک ۰ تا ۹ را با تعداد مرتب شده به صورت کاوهشی همراه با تعداد در خروجی برگرداند. خروجی این کوئری به صورت زیر است.

17.0	نیوروز 1401 :
9.0	سفرهای نیوروزی :
7.0	کرونیا در اینده :
7.0	بیلیس راهور :
5.0	همه-نیام-علیه-کرونیا :
5.0	شهرداری تهران :
4.0	کرونیا و پریوس :
4.0	پیک ششم کرونیا :
4.0	ویبرومن کرونیا :
4.0	شهر تهران :

برای پاسخ به کوئری سوم، از تعدادی لیست استفاده کردیم. نام هم لیست با فرمت day-month مشخص می شود و کد هر خبر را در تاریخ مربوطه push می کنیم. در نتیجه برای هر روز یک لیست خواهیم داشت که کد خبرهای آن روز در آنها قرار دارد. می توانستیم برای هر هفته یک list داشته باشیم، اما از آنجایی که ممکن است یک نفر بخواهد خبرها از سه شنبه تا سه شنبه را انتخاب کند، اگر برای هر روز یک لیست داشته باشیم انعطاف بیشتری خواهیم داشت. برای شمارش تعداد خبرهای منتشر شده در هفته‌ی ۲۱ تا ۲۷ اسفند، با دستور Llen طول لیست‌هایی با نام ۱۲-۲۱، ۱۲-۲۲، ۲۱-۱۲ را بدست می‌آوریم و طول آنها را با هم جمع می‌کنیم. مجموع خبرهای منتشر شده در این تاریخ‌ها برابر با ۱۸۵۸ خبر بود.

همچنین برای کوئری چهارم هم، برای هر روز، یک ست با نام day-month-category ایجاد کردیم که عنوان خبرها را در آنها add می‌کنیم. نهایتا برای به دست آوردن عنوان خبرهای اقتصادی در یک روز خاص، با smembers یک ست با نام مشخص شده می‌توانیم آنها را دریافت کنیم. عنوان بخشی از خبرهای اقتصادی منتشر شده در ۲۷ اسفند به صورت زیر است.

سهمیه پلزین نیروزی مبنی‌است/ می‌دم از ذخایر ۹ ماهه پیوخت خود استفاده کنند	تکمیل رجیستری برای ساخت هوشیند و نیابت
سقف جدید سیماه سیخایی‌ها برای امتحان جنیفی افزایش می‌باشد	واردات کالای اسلامی ۱۸ میلیارد دلاری شد
چند سهم سیخایی‌ها قیوخته شد؟	سیماهی «درگاه ملی درخواست مجوز جایگاه پیوخت» راه‌اندازی شد
امتناع او آتش پریوس کمپرسکی در اینتلیا محدود می‌شود	امتناع او آتش پریوس کمپرسکی در اینتلیا محدود می‌شود
پریوس و فیلتر شناور چوده و نازدیک صفت مینیموم شد	قیمت سکه و طلا امسال رشد کمی داشت/کاهش ایوان سنه پس از تعطیلهای نیروز
از پیغام پیارانه ۱۵۰ هزار یورویی با همه عبارت	افزایش ۲ هزار میلیون برق با عدم تغییر رسمی میانه کشور
ایرانی پیروزانها در فرودگاه‌ها اقام خدمت ارتقا پذیرفت	پیارهای نیروزی سبل آما نیستند/ وضعیت دما معتدل و چیزی که دوچرخه پیشینی نمی‌کند
اطلاعات مشترکان کم می‌رفت پوشش نهادهای جملیتی بیرون شناسه دریافت شد	پیش‌بینی ۲۰ سیاره چادر غایری برای استقرار از اسفاران نیروزی
وضعیت پیش‌بینی کارگران در ۱۴۰۱	خطوط هوایی حق فروش پلیت با افزایش قیمت را تقدیر
هدف دولت پیزدید رشد اقتصادی فراکیر است/ در مذاکرات برای هر سیاره‌یی آغازه ایم	هدف دولت پیزدید رشد اقتصادی فراکیر است/ در مذاکرات برای هر سیاره‌یی آغازه ایم
افزایش نرخ پیوه در افغانستان ادامه دارد؟	افزایش نرخ پیوه در افغانستان ادامه دارد؟
انفاق کم‌نظیر پیزارد سکه و طلا در آخرين میله‌ی قرن	انفاق کم‌نظیر پیزارد سکه و طلا در آخرين میله‌ی قرن
راهبرد متفقی اقتصاد مقاومتی فراموش نیشود	راهبرد متفقی اقتصاد مقاومتی فراموش نیشود
نمود فعالیت	نمود فعالیت
چین دومنی اقتصاد جهان و اولین شریک تجاری ایران است/ پاییز به سمت کیاوش وایمکی به نفت جرکیه کلیم	چین دومنی اقتصاد جهان و اولین شریک تجاری ایران است/ پاییز به سمت کیاوش وایمکی به نفت جرکیه کلیم
کامهای جدید تجارت ایران و ترکیه هدیه و پیش ایرانسل برای عین نیمه شبیان اعلام شد	کامهای جدید تجارت ایران و ترکیه هدیه و پیش ایرانسل برای عین نیمه شبیان اعلام شد

برای پیاده‌سازی حافظه‌ی نهان، کد خبرها را به خلاصه‌ی آن‌ها مپ می‌کنیم. این کار برای این است که تنها با `get` کردن کد یک خبر، به نتیجه برسیم. اما با این روش نمی‌توانیم از تعداد خبرهای موجود در `cache` به سادگی مطلع شویم. به این منظور از یک استفاده می‌کنیم. زمانی که یک کد در `cache` وجود ندارد، در صورتی که طول لیست ما کمتر از ۱۰ باشد، کد جدید را در آن `lpush` می‌کنیم و `key-value` کد خبر-خلاصه را هم در `redis` اضافه می‌کنیم. در صورتی که اندازه‌ی `cache` برابر با ۱۰ باشد، یک مقدار از لیست را `rpop` می‌کنیم، رکورد متناظر آیدی پاپ شده را از `redis` حذف می‌کنیم، و کد جدید را `lpush` کرده و `key-value` کد خبر-خلاصه متناظر را در `redis` اضافه می‌کنیم.

پیاده‌سازی حافظه‌ی نهان با سیاست زمانی بسیار ساده‌تر است. هر خبری که دیده می‌شود، با دستور `set`، مقدار کد خبر را به خلاصه‌ی آن مپ می‌کنیم و مقدار `ttl` آن را با استفاده از آرگومان `ex` برابر با ۱۰ ثانیه قرار می‌دهیم تا بعد از گذشت ۱۰ ثانیه از ردیس حذف شود. این بار نیازی به چک کردن وجود یا عدم وجود `key` نیست، چرا که وقتی یک `key` که همچنان وجود دارد، دوباره می‌رسد، مقدار `ttl` آن باید تمدید شود.

سوال ۴-

برای بالا آوردن cassandra از ایمیج داکر آن استفاده کردیم و فایل docker-compose به صورت زیر است:

```
services:
  cassandra:
    image: cassandra
    ports:
      - 9042:9042
    volumes:
      - cassandra:/var/lib/cassandra
    environment:
      - CASSANDRA_CLUSTER_NAME=sahar

volumes:
  cassandra:|
```

همچنین برای نوشتن داده‌ها و ایجاد جدول‌ها در driver-Cassandra هم از cassandra در پایتون استفاده کردیم.

(الف)

- تفاوت جداول static و dynamic

جدول‌های static، یک سرت از ستون‌ها دارند و خیلی شبیه به relational-database هاستند. جداول dynamic به ما این اجازه را می‌دهند که به صورت واضح و مشخصی تمامی column‌ها را مشخص نکرده و اجازه‌ی داشتن ستون‌های دلخواه را به ما می‌دهد.

- الگوریتم‌های replication

دو الگوریتم replication در cassandra وجود دارد.

۱- simple strategy که در یک دیتابیس‌تر واحد و یک rack استفاده می‌شود و در صورتی که چند دیتابیس‌تر داشته باشیم نمی‌توان از آن استفاده کرد. در این الگوریتم اولین replica در نودی که partitioner مشخص می‌کند قرار می‌گیرد و replica‌های بعدی، در نود بعدی کلاستر در جهت چرخش ساعت قرار می‌گیرد بدون در نظر گرفتن توپولوژی (چون بر روی یک rack و دیتابیس‌تر است).

۲- network topology strategy که استراتژی پیشنهاد شده است به علت اینکه برای اسکیل کردن بسیار مناسب است و زمانی که چند دیتابیس‌تر داریم استفاده می‌شود. باید مشخص کنیم که چند replica در هر دیتابیس‌تر نیاز داریم. این الگوریتم replica در یک کلاستر را در جهت عقربه‌های ساعت حرکت می‌کند تا به زمانی که به اولین نود در یک rack دیگر برسد. این استراتژی سعی می‌کند که replica‌ها را در rack‌های مختلف قرار بدهد؛ چرا که احتمال رخداد خطای همزمان در نودهای روی یک rack بسیار بیشتر است.

ب)

از آنجایی که در cassandra امکان join و يا sort نداریم، در صورتی که به این دو عمل نیاز داشته باشیم، این اعمال باید قبل از نوشته شدن داده در Cassandra صورت بگیرند. از طرفی-partition key محل قرار گرفتن داده در کلاستر را مشخص می کند و cluster-key مشخص می کند که داده، در هر partition با چه ترتیبی قرار بگیرد. با توجه به این نکات، ما در ادامه طراحی جدول ها برای هم کوئری را بررسی می کنیم و چنان که نیاز به دیتای دو جدول متفاوت از دیتابیس داده شده داشته باشیم، قبل از نوشتن در cassandra آن ها را جوین می کنیم.

فایل نوشتن داده‌ها در cassandra در فایل‌هایی همان‌نام با فایل کوئری آن‌ها در پوشه‌ی python-
codes قرار دارد.

- نام فیلم‌هایی که در ژانر درام هستند، به ترتیب امتیاز imdb

برای این کوئری جدولی طراحی می‌کنیم که partition-key آن، ژانر باشد و cluster-key آن هم امتیاز imdb باشد. این جدول باید شامل نام فیلم باشد. این داده‌ها در دو فایل قرار دارد. فایل title-rating آیدی فیلم و امتیاز هر فیلم را دارد و فایل title-basics آیدی فیلم و title و ژانر آن را دارد. به این منظور برای نوشتن داده در جدول cassandra از هر دوی این فایل‌ها استفاده می‌کنیم و با استفاده از فیلد مشترک آیدی فیلم، عنوان، ژانر و امتیاز هر فیلم را بدست می‌آوریم و در جدول می‌نویسیم.

البته از آنجایی که هر فیلم ممکن است چندین ژانر داشته باشد، و ما امکان سرچ کردن با regex و غیره را نداریم و همچنین تایپ‌هایی مانند list نمی‌توانند در primary-key وجود داشته باشند، ما به ازای هر ژانر در هر فیلم یک رکورد مجزا ایجاد کردیم. یعنی در صورتی که یک فیلم ژانر درام، ترسناک داشته باشد، یک رکورد با ژانر درام، و یک رکورد با ژانر ترسناک برای آن فیلم ایجاد می‌شود.

کوئری نهایی به این صورت است که title و امتیاز imdb را در عبارت select ای که با شرط = 'Drama' محدود شده برمی‌گردانیم و از آنجایی که rating در واقع cluster-key هم هست، نتایج به ترتیب امتیاز imdb مرتب می‌شوند (می‌شود در هنگام ساخت جدول مشخص کرد که cluster-key با ترتیب صعودی، مرتباً کند و باز نول.).

title	rating
Zarabanda	10
Your Death, my Life	10
Yes He Can!	10
Whoa	10
What I Can't Tell You	10
We're The Black Girls	10
Virtual Love in Lockdown	10
Vijay Questions Pallavi	10
Unlucky Lavender	10
Unique Chance	10
Under Pressure	10
Uhinchaledu Kadu	10
Tu Saath hai Na	10
Thurje # Hashtag	10
Theatrical Dilemma	10
The Virus Hits Home	10
The Terrorist	10
The Sabharwals Are Here	10
The Praetorian	10
The Planting of Trees	10

بخش ابتدایی نتیجه‌ی کوئری به صورت زیر است:

- فیلم‌هایی که در آن‌ها Steven Spielberg کارگردان و نویسنده بوده است.

برای اینکه بتوانیم با یک تساوی، داده را فیلتر کنیم، فیلدهای مورد نیاز برای فیلتر باید در partition-key باشند. به همین منظور برای این کوئری یک جدول ایجاد کردیم که نویسنده و کارگردان در آن به عنوان partition-key هستند. البته مشابه قسمت قبل، از آنجایی که یک فیلم می‌تواند چند نویسنده و چند کارگردان داشته باشد، ما در هر کدام از این دو فیلد، به ازای هر فرد یک رکورد در نظر گرفتیم. یعنی در صورتی که یک فیلم ۲ کارگردان و ۲ نویسنده داشته باشد، ما ۴ رکورد برای ۴ ترکیب مختلف از این ۴ نام در جدول خود خواهیم داشت. همچنین این جدول باید عنوان فیلم را هم داشته باشد.

برای ساخت این جدول باید ابتدا از فایل name-basic مپینگ ایدی افراد به نام آن‌ها را استخراج می‌کردیم. سپس با title-basics که آیدی فیلم و عنوان آن را دارد، و title-crew که آیدی فیلم و آیدی نویسنده‌گان و کارگردانان فیلم را دارد استفاده از فیلد مشترک آیدی فیلم، برای هر فیلم، عنوان، آیدی کارگردانان و آیدی نویسنده‌گان را مشخص می‌کنیم. سپس با استفاده از مپینگی که در بخش قبل از آیدی دست‌اندرکاران ایجاد کردیم، ترکیب عنوان فیلم، نام کارگردانان و نام نویسنده‌گان را ایجاد می‌کنیم و مطابق توضیحات در جدول می‌نویسیم.

برای زدن کوئری هم باید در شرط where نویسنده و کارگردان را مساوی با اسم گفته‌شده قرار دهیم و شرط را با همان ترتیبی بنویسیم که در partition-key مشخص شده است.

نتیجه کوئری در تصویر زیر مشخص است.

title	writer	director
The Fabelmans	Steven Spielberg	Steven Spielberg

- فیلم‌های با امتیاز بیشتر از ۷ که در سال ۲۰۲۱ منتشر شده‌اند به ترتیب نزولی تعداد آرا.

فرض می‌کنیم که با استفاده از این جدول، بخواهیم فیلم‌هایی را در سال‌های مختلف، همراه با رنچ‌های مختلفی از امتیاز مرتب کنیم. برای این کار، به رنچ‌های مختلف امتیاز، رنک‌های مختلفی نسبت می‌دهیم. برای مثال در این سوال رنک امتیازی فیلم‌هایی که امتیاز بیشتر از ۷ داشته‌اند، برابر با ۱ است.

جدولی طراحی می‌کنیم که در آن سال تولید و رنک امتیازی، partition-key و تعداد آراء دریافتی به ترتیب نزولی، cluster-key باشند. این جدول همچنین باید عنوان فیلم و امتیاز اصلی فیلم را هم داشته باشد.

برای ساخت این جدول، از فایل title-basic که آیدی فیلم و عنوان و سال تولید را دارد استفاده می‌کنیم و از فایل title-rating هم که آیدی و امتیاز فیلم و تعداد آرا را دارد استفاده می‌کنیم و با ترکیب اطلاعات این دو فایل، مجموعه‌ی عنوان فیلم، سال تولید، امتیاز و تعداد آرا را برای هر فیلم به دست می‌آوریم. سپس برای نوشتن در جدول، با توجه به امتیاز فیلم، رنک امتیازی فیلم را هم مشخص می‌کنیم و در جدول می‌نویسیم. سپس برای نوشتن کوئری، شرط where را با توجه به سال ساخت فیلم

و رنک امتیازی مشخص می‌کنیم و از آنجایی که تعداد آرا به عنوان کلاستر-کی بوده است، نتایج به ترتیب نزولی این مقدار مرتب خواهد بود.

قسمت اول نتیجه‌ی کوئری در شکل زیر مشخص است:

title	rating	vote_count
Spider-Man: No Way Home	8.5	566982
Dune: Part One	8.1	531729
Don't Look Up	7.2	477334
Ojing-eo geim	8	411439
Zack Snyder's Justice League	8.1	369445
Shang-Chi and the Legend of the Ten Rings	7.5	340089
No Time to Die	7.3	333930
Free Guy	7.2	320594
The Suicide Squad	7.2	318293
Aspirants	9.7	292432
WandaVision	8	275664
Loki	8.3	273920
Nobody	7.4	216749
Cruella	7.4	216327
The Falcon and the Winter Soldier	7.3	193785
Jai Bhim	9.4	184342
Encanto	7.3	177375
Arcane: League of Legends	9.1	168169
Wrath of Man	7.1	155524
Luca	7.5	145412

- نام کارگردان‌ها و نویسندها فیلم‌های کوتاه.

برای این کوئری یک جدول درست می‌کنیم که در آن ژانر فیلم partition-key است که در جدول نام کارگردان‌ها، نویسندها و عنوان خود فیلم‌ها وجود دارد. برای ساخت این جدول، از فایل title-basics که آیدی فیلم و عنوان فیلم و ژانر را دارد و فایل title-crew که آیدی فیلم و آیدی نویسندها و کارگردان‌های هر فیلم را دارد استفاده می‌کنیم. همچنین توسط فایل name-basics آیدی name-basics مبینگ آیدی عوامل به نام آن‌ها را به دست می‌آوریم و مجموعه‌ی عنوان فیلم، ژانر، نام نویسندها و نام کارگردان‌ها تبدیل می‌کنیم.

همچنین از آنجایی که هر فیلم چندین ژانر دارد، مانند توضیحات قسمت‌های پیشین، به ازای هر ژانر، یک رکورد در جدول ایجاد می‌کنیم.

برای کوئری زدن، کافی است تنها شرط where را برابر با ژانر مورد نظر قرار بدهیم و عنوان فیلم و نام نویسندها و کارگردان‌ها را برگردانیم.

بخشی از نتیجه‌ی کوئری به صورت زیر است:

title	genre	writers	directors
#1	Short	Evan Materne	Evan Materne
#1 Dad	Short	John M. Hoffman	Ralph Smith
#1 Fan	Short	Livia Rae	Livia Rae
#2young	Short	Aaron Toronto-Jordan Toronto-Matthew Toronto	Matthew Toronto
#6 Feet Apart	Short	Austance Caroline-Lea Wülfert	Lea Wülfert
#67times	Short	Jane Slavin	Jane Slavin
#APH2 Labels	Short	Indrakarma Baishnawb	Indrakarma Baishnawb-A.M. Naveen Kumar
#Anniversary	Short	Ellis E. Fowler	Ellis E. Fowler
#BestSelfieEver	Short	Inti Carrizo-Ortiz	Inti Carrizo-Ortiz
#BlueBoar	Short	Alice Victoria Winslow	Alice Victoria Winslow

- ستونی برای تعداد نسخه‌های فروخته شده اضافه کنید و عدد ۱۰۰۰ را برای حداقل تعداد نسخه برای فیلم‌های سال ۲۰۲۱ به قبل اضافه کنید.
برای اضافه کردن یک فیلد به داده، ابتدا باید با دستور

`ALTER TABLE <table_name> ADD <fileld_name> <data_type>`

این فیلد را به schema اضافه کنیم.

برای اینکه بتوانیم این دستور را انجام دهیم، از جدولی استفاده می‌کنیم که partition-key آن، سال انتشار آن باشد. برای update یک فیلد در Cassandra باید در بخش where تمامی مشخصات مربوط به primary-key آنها را مشخص کرده باشیم. از طرفی key-primary مربوط به هر فیلم، باید یا شامل نام فیلم و یا آیدی فیلم باشد و بقیه‌ی فیلدها لزوماً یکتا نیستند. لذا به نظر می‌رسد راهی برای افروzen تنها از طریق cql و با فیلتر بر روی سال، وجود ندارد. اما می‌توانیم در سطح اپلیکیشن این کار را انجام دهیم.

(ج)

- متوسط امتیاز فیلم‌ها در هر یک از ژانرهای.

برای این بخش از جدول طراحی شده در کوئری اول قسمت ب استفاده شده است که partition-key آن همان ژانر فیلم است. به این منظور، ما داده را بر روی ژانر group by می‌کنیم، سپس عبارت sum(rating)/count(*) را برای هر دسته محاسبه می‌کنیم، بخشی از خروجی کوئری به صورت زیر است:

genre	avg_rating
undefined	6.88158
Musical	7.40877
Family	7.1048
Comedy	7.10082
News	6.60454
Game-Show	7.12194
Short	7.30483
Action	7.32568
Music	7.2449
Biography	7.40536
War	7.6402
Adult	7.25106
Horror	6.47482
Mystery	7.25062
Reality-TV	7.01456
Sport	6.81482
Drama	7.20403
Romance	7.00679
Documentary	7.38801
Thriller	6.76652
Crime	7.26684

- تعداد فیلم‌های هر یک از کارگردان‌ها در سال‌های ۲۰۲۱ و ۲۰۲۲.

برای اینکه بتوانیم تعداد فیلم‌های هر کارگردان را بشماریم، باید داده را با توجه به کارگردان group by کنیم. برای این کار جدولی ساختیم که کارگردان partition-key آن است و سال تولید فیلم

count	director
47	Marc Martin
1	Christian Baumeister
1	Jared Walker
1	Aaron Geva
1	Rebecca Sugar
3	Francois Reinhardt
1	Jeffrey Arsenault
2	Karina Lafayette
1	Mesaj
1	Easy Matsumoto
1	Cal Murphy Barton
1	Pascal Schuh
4	Tim Leandro
1	Brittany Samson
6	Merlin Crossingham
1	Christine Kilmer
1	Henrike Kull
1	Nitin G.
10	Yaron Shilon
2	Rakib Erick
1	Katharine Phillips
3	Pierre Joseph
1	Kena Sosa
2	Gabriel Bocanegra
5	Laura Way
1	Luca Mandrile
4	Kevin Hartford
3	William Lilly
1	Tarnvir Singh Jagpal
1	Jordan Sarf
1	Ryon Lee

برای همین با استفاده از دو فایل title-crew و title-basics که اولی آیدی فیلم، عنوان و سال پخش آن را دارد و دومی آیدی فیلم و نام کارگردان را دارد استفاده کردیم تا این داده‌ها را join کنیم. همچنین از فایل name-basics هم استفاده می‌کنیم که مپینگ آیدی افراد به اسم آن‌ها را داشته باشیم. برای انجام این کوئری از group by director استفاده کردیم و سال‌های بزرگ‌تر از ۲۰۲۰ را در نظر گرفتیم. همچنین این کوئری نیاز به استفاده از کلیدواژه‌ی ALLOW FILTERING دارد اما با توجه به طراحی جدول، به نظر نمی‌رسد که performance دچار نقص جدی بشود. بخشی از نتیجه‌ی این کوئری در تصویر زیر قابل ملاحظه است.

- ۱۰ فیلم پربازدید سال‌های ۲۰۲۱ و ۲۰۲۰ بر مبنای تعداد آرا ثبت شده.

title	publish_year	vote_count
Tenet	2020	460913
The Queen's Gambit	2020	399441
Soul	2020	304931
Wonder Woman 1984	2020	251689
Birds of Prey: And the Fantabulous Emancipation of One Harley Quinn	2020	225878
The Invisible Man	2020	213365
A Quiet Place Part II	2020	202575
Extraction	2020	190349
Ted Lasso	2020	173574
The Trial of the Chicago 7	2020	169416
Spider-Man: No Way Home	2021	566982
Dune: Part One	2021	531729
Don't Look Up	2021	477334
Ojing-eo geim	2021	411439
Zack Snyder's Justice League	2021	369445
Black Widow	2021	344710
Shang-Chi and the Legend of the Ten Rings	2021	340089
No Time to Die	2021	333930
Free Guy	2021	320594
The Suicide Squad	2021	318293

برای این بخش از جدولی استفاده کردیم که partition-key آن سال پخش و cluster-key آن تعداد آرای هر فیلم باشد. همچنین این جدول شامل نام فیلم هم هست. برای ساخت این جدول، از فایل title-rating استفاده کردیم که اولی شامل آیدی فیلم، نام فیلم و سال انتشار است و دومی basics شامل آیدی فیلم و تعداد رای‌ها. بعد از انجام join روی این دو جدول به صورت پیش‌فرض آن‌ها را در دیتابیس طراحی شده ریختیم. برای کوئری، سال انتشار را محدود به سال‌های ۲۰۲۰ و ۲۰۲۱ کردیم و

لیمیتی برابر با ۱۰ رکورد در هر پارتیشن قرار دادیم. البته داده‌ها در cassandra در هر پارتیشن سورت هستند و برای اینکه کل داده‌ها را سورت کنیم می‌توانیم در سطح اپلیکیشن اقدام کنیم.

- متوسط زمان فیلم‌های هر یک از دسته‌بندی‌ها (کوتاه، سریال تلویزیونی و ...) که به ترتیب صعودی مرتب شده است.

برای این بخش جدولی طراحی کردیم که partition-key آن ژانر و primary-key آن مدت‌زمان فیلم است. همچنین به ترکیب این دو title هم برای اضافه شده است. برای ساخت این جدول از فایل title-basics استفاده کردیم که همه‌ی مقادیر مورد نیاز را دارد. برای کوئری، از group by genre استفاده می‌کنیم و میانگین مدت‌زمان اجرا را برای فیلم‌ها به دست می‌آوریم (به ازای هر ژانر از یک فیلم، یک رکورد در دیتابیس وجود دارد) البته این نتیجه بر اساس این میانگین سورت نشده و Cassandra هم این امکان را به ما نمی‌دهد. در نتیجه ما اجبارا باید از یک جدول میانی برای انجام کار خود استفاده کنیم. برای این کار قبل از اجرای کوئری از دستوری 'capture temp.csv' استفاده می‌کنیم و بعد از اجرا از دستور off capture temp.csv. این کار باعث می‌شود تا نتیجه‌ی کوئری در فایل temp.csv ذخیره شود. با تغییراتی در فایل temp.csv آن را به فرمت استاندارد csv می‌بریم، همچنین یک فیلد aim با عنوان sort به هر سطر اضافه می‌کنیم تا داده‌ها را با این مقدار partition کنیم و بتوانیم داده‌ها که اجبارا در یک partition قرار می‌گیرند را سورت کنیم. به این منظور یک table تعريف می‌کنیم که آن همان فیلد aim است که برای تمامی خطوط مقدار یکسان دارد و partition-key primary-key می‌گذاریم تا داده بر اساس آن سورت شود. همچنین فیلد genre هم در avg_min می‌حضور دارد. نهایتا مقادیر genre و avg_min را از این جدول دریافت می‌کنیم که نتیجه برابر عکس زیر است که بر اساس میانگین به صورت صعودی مرتب شده‌است.

genre		avg_min
	Short	11
Animation		19
	Comedy	34
	Family	34
	Sci-Fi	34
	Music	36
	Horror	39
	Drama	43
Adventure		45
Documentary		46
	Fantasy	46
	Musical	46
	Romance	47
	Action	48
	Crime	49
	Mystery	50
	Thriller	51
	Game-Show	52
	History	52
Reality-TV		52
	New	57
Biography		59
Talk-Show		59
	War	61
	Adult	65
	Sport	85

- تعداد فیلم‌های سال ۲۰۲۰ در ژانرهای مستند، تاریخی و Reality-TV

برای این بخش، یک جدول طراحی کردیم که genre و publish-year هر دو partition-key هستند. برای این کار تنها به فایل title-basic نیاز داریم. زیرا هر دوی این اطلاعات در این فایل وجود دارد. برای کوئری، سال را محدود به ۲۰۲۰ می‌کنیم و از دستور in برای محدود کردن ژانرهای استفاده می‌کنیم. سپس بر روی ژانر group-by می‌کنیم و تعداد را با استفاده از count(*) بدست می‌آوریم. نتیجه کوئری به صورت زیر است.

count_	genre	publish_year
32844	Documentary	2020
5214	History	2020
19960	Reality-TV	2020

(د)

برای مشاهده write latency و read latency در هر جدول، از دستور

Nodetool tablehistograms keyspace table

استفاده می‌کنیم. در زیر ابتدا نتیجه را برای هر جدول مشاهده می‌کنید و در نهایت مقایسه را انجام می‌دهیم.

برای جدول most-visited که برای بخش سوم قسمت ج استفاده شده است:

Percentile	Read Latency (micros)	Write Latency (micros)	SSTables	Partition Size (bytes)	Cell Count
50%	2816.16	17.08	0.00	NaN	NaN
75%	2816.16	24.60	0.00	NaN	NaN
95%	2816.16	42.51	0.00	NaN	NaN
98%	2816.16	61.21	0.00	NaN	NaN
99%	2816.16	88.15	0.00	NaN	NaN
Min	2346.80	3.97	0.00	NaN	NaN
Max	2816.16	654.95	0.00	NaN	NaN

برای جدول rating_tables که در کوئری اول قسمت ج استفاده شده است:

Percentile	Read Latency (micros)	Write Latency (micros)	SSTables	Partition Size (bytes)	Cell Count
50%	2346.80	24.60	0.00	NaN	NaN
75%	2346.80	35.43	0.00	NaN	NaN
95%	2346.80	61.21	0.00	NaN	NaN
98%	2346.80	88.15	0.00	NaN	NaN
99%	2346.80	105.78	0.00	NaN	NaN
Min	1955.67	4.77	0.00	NaN	NaN
Max	2346.80	219.34	0.00	NaN	NaN

برای جدول dir_pub که در کوئری دوم استفاده شده است:

Percentile	Read Latency (micros)	Write Latency (micros)	SSTables	Partition Size (bytes)	Cell Count
50%	943.13	11.86	0.00	NaN	NaN
75%	943.13	20.50	0.00	NaN	NaN
95%	943.13	42.51	0.00	NaN	NaN
98%	943.13	61.21	0.00	NaN	NaN
99%	943.13	73.46	0.00	NaN	NaN
Min	785.94	2.76	0.00	NaN	NaN
Max	943.13	12108.97	0.00	NaN	NaN

برای جدول genre_min استفاده شده در کوئری چهارم بخش ج:

Percentile	Read Latency (micros)	Write Latency (micros)	SSTables	Partition Size (bytes)	Cell Count
50%	1358.10	11.86	0.00	NaN	NaN
75%	1358.10	17.08	0.00	NaN	NaN
95%	1358.10	35.43	0.00	NaN	NaN
98%	1358.10	61.21	0.00	NaN	NaN
99%	1358.10	105.78	0.00	NaN	NaN
Min	1131.75	1.92	0.00	NaN	NaN
Max	1358.10	3379.39	0.00	NaN	NaN

برای جدول year_genre استفاده شده در کوئری پنجم:

Percentile	Read Latency (micros)	Write Latency (micros)	SSTables	Partition Size (bytes)	Cell Count
50%	545.79	14.24	0.00	NaN	NaN
75%	545.79	20.50	0.00	NaN	NaN
95%	545.79	35.43	0.00	NaN	NaN
98%	545.79	51.01	0.00	NaN	NaN
99%	545.79	73.46	0.00	NaN	NaN
Min	454.83	2.76	0.00	NaN	NaN
Max	545.79	943.13	0.00	NaN	NaN

همانطور که مشخص است، به صورت میانگین در این جداول read-latency بسیار بیشتر از write-latency داریم. این دلیلی است که دیتابیس Cassandra را برای موقعی که rate نوشتن بیشتر از خواندن است، دیتابیس مناسبی می‌کند.

سوال ۵-

فایل docker-compose برای بالا آوردن neo4j به صورت زیر است.

```
services:  
  neo4j:  
    image: neo4j  
    hostname: neo4j  
    container_name: neo4j  
    ports:  
      - "7474:7474"  
      - "7687:7687"  
    volumes:  
      - neo4j:/plugins  
    environment:  
      NEO4J_AUTH: neo4jstreams  
      NEO4J_dbms_logs_debug_level: DEBUG  
      NEO4J_streams_source_topic_nodes_neo4j: Person{*}  
      NEO4J_streams_source_topic_relationships_neo4j: KNOWS{*}  
      NEO4JLABS_PLUGINS: '[ "apoc" ]'  
      NEO4J_apoc_export_file_enabled: 'true'  
  
volumes:  
  neo4j:
```

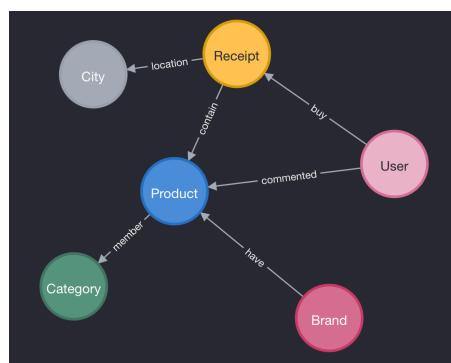
افزونه‌ی apoc برای خروجی گرفتن در csv هم نیاز بوده است که اضافه شده است.

(الف)

در این دیتابیست، ۵ فایل وجود دارد، فایل users که اطلاعات کاربران را دارد، فایل product اطلاعات محصولات را دارد، فایل orders نشان می‌دهد هر کاربر چه سفارش‌هایی داشته است (که شامل آیتم‌های سفارشی هم و مبلغ سفارش هم هست)، فایل keifiat نظرات کاربران برای محصولات مختلف را شامل می‌شود و tarikhche kharid نشان می‌دهد که هر محصول با چه قیمت‌های فروخته شده است.

در دیتابیس طراحی شده، ما نودهای User، Product، Category، Brand، City، Receipt را داریم که Receipt از contain، City از location، Receipt از buy، User از commented، Product از member و Product از have به Product از have، Brand به User از commented، Product به User از commented، Product به Category تعريف شده‌اند.

شمای کلی پایگاه داده، به شکل زیر است.



در ادامه‌ی نحوه‌ی ساخت هر نود و relation را توضیح می‌دهیم.

:User نود

با پیمایش فایل users، به ازای هر user یک نود با آیدی، نام، سن، جنسیت، ایمیل و یوزنیم هر یوزر ایجاد کردیم.

:Product نود

با پیمایش فایل products، به ازای هر product، یک نود با عنوان‌های فارسی و انگلیسی، کد url، و ویژگی‌های پروداکت ایجاد کردیم.

:Brand نود

با پیمایش فایل products، نام‌های یکتا در ستون brand_name_en را پیدا کردیم، و به ازای هر کدام یک نود که شامل نام برنده است ایجاد کردیم.

:Category نود

مجدداً با پیمایش فایل products، مقادیر یکتا در ستون category_title_fa را به دست آوردیم و به ازای هر کدام یک نود شامل نام کتگوری ایجاد کردیم.

:Receipt نود

با پیمایش فایل orders، به ازای هر سفارش یک نود با آیدی سفارش، مبلغ سفارش و تاریخ سفارش ایجاد کردیم.

:City نود

برای ایجاد نود city هم با پیمایش فایل orders، مقادیر یکتا در ستون city_name_fa را به دست آوردیم و به ازای هر کدام یک نود city که شامل نام آن شهر است ایجاد کردیم.

:buy رابطه‌ی

در فایل orders، در هر خط ما آیدی مشتری، آیدی آیتم و اطلاعات دیگری را داریم. با دریافت آیدی خود سفارش و آیدی مشتری و استفاده از دو matcher، یکی برای Users و یکی برای Receipts نود user و receipt مربوط به هر خط را بدست آوردیم و سپس یک رابطه از آن یوزر به سمت آن رسید با نام buy ایجاد می‌کنیم.

:contain رابطه‌ی

هر order شامل آیدی محصول هم هست. این بار با استفاده از همان فایل orders و دو matcher، یکی برای product و یکی برای receipt، نود معادل محصول و رسید را برای هر رکورد پیدا کردیم و یک رابطه به نام contain از نود رسید به نود محصول ایجاد کردیم.

:location رابطه‌ی

هر order به مقصد یک شهر ثبت می‌شود. در پیمایش فایل orders با استفاده از نام شهر، نود مربوط به شهر را پیدا می‌کنیم و یک رابطه از نود رسید مربوط به آن رکورد به نام location با سمت شهر پیدا شده ایجاد می‌کنیم.

:member رابطه‌ی

هر محصول، عضو یک دسته است. برای ساخت این رابطه، با پیمایش فایل products، با استفاده از product و category نود را با استفاده از id محصول و نود کتگوری را با استفاده از نام کتگوری پیدا می‌کنیم و رابطه‌ی عضویت را از نود محصول به سمت یک کتگوری با نام member ایجاد می‌کنیم.

:have رابطه‌ی

هر برنده، شامل تعدادی محصول است. در پیمایش فایل products، با استفاده از matcher محصول و آیدی محصول، نود مربوط به آن محصول را پیدا می‌کنیم. همچنین با استفاده از نام brand، نود مربوط به را هم پیدا می‌کنیم و رابطه‌ی have را از نود برنده به سمت نود محصول، ایجاد می‌کنیم.

:commented رابطه‌ی

هر کاربر می‌تواند برای محصولات مختلف، نظر داشته باشد. برای ایجاد این رابطه، در حین پیمایش فایل keifiyat، با استفاده از دو matcher یکی برای کاربر و یکی برای محصول، و به کمک ستون‌های product_id و user_id، نود مربوط به محصول و کاربر را پیدا می‌کنیم و رابطه‌ی commented را از نود کاربر به سمت نود محصول با ویژگی score که از ستون score فایل بدست می‌آید ایجاد می‌کنیم.

برای اضافه کردن این اطلاعات به دیتابیس، از لایبرری مربوطه در زبان python استفاده شده است و کد آن هم در کنار گزارش آپلود شده است.

(ب)

کوئری اول:

برای این کوئری از رابطه‌ی buy استفاده کرده‌ایم و تعداد رابطه‌های خروجی از هر user را شمارش کردیم و هیچ یوزری بیش از یک سفارش نداشت. در نتیجه اولین user را در خروجی ظاهر کردیم.

```
neo4j$ match (u:User)-[r:buy]→() Return u, count(r) as order_count order by order_count desc limit 1;
```

"u"	"order_count"
{"gender": "male", "name": "رسین گلشن", "id": 4051236, "email": "arsyn.glsn1hn@example.com", "age": 51, "username": "goldenelephant418"}	1

برای پیدا کردن user با بیشترین مبلغ سفارش هم مشابه قبل عمل کردیم و این بار از sum بر روی مبلغ سفارش استفاده کردیم. خروجی در زیر قابل مشاهده است.

```
match (u:User)→(r:Receipt) Return u, sum(r.amount) as amount order by amount desc limit 1;
```

"u"	"amount"
{"gender": "female", "name": "ملینا احمدی", "id": 5400422, "age": 28, "email": "3288716.0", "username": "heavywolf954"}	3288716.0

کوئری دوم:

برای این کوئری از رابطه‌ی location استفاده کردیم. و نام‌های یکتا در نودهای city که در فیلتری که بر روی تاریخ اعمال شده است وجود داشته‌اند را در خروجی برگردانده‌ایم. برای فیلتر تاریخ، فیلد تاریخ سفارش را بر روی «-» قسمت کردایم و قسمت دوم ماه می‌باشد و شرط ما این است که سفارش در ماه دوم سال انجام شده باشد. نتیجه‌ی کوئری به صورت زیر است.

```
match (r:Receipt)-[l:location]-(c:City) where [item in split(r.date, '-')][1] = '02' return distinct c.name;
```

c.name
"تهران"
"مشهد"
" Tehran"
"Tehran"
"تهران"
"Tehran"

کوئری سوم:

برای این کوئری از دو رابطه‌ی have و commented استفاده کردیم. در واقع کاربران برای محصولات نظر ثبت می‌کنند و هر نظر متعلق به یک brand است و با ترکیب آن‌ها در واقع join انجام داده‌ایم. در نهایت هم نام هر برند را به همراه متوسط امتیاز ثبت شده در خروجی برگردانده‌ایم. نتیجه به این صورت

```
neo4j$ match (b:Brand)-[l:have]-(p:Product)-[c:commented]-(u:User) return avg(c.score), b.name
```

avg(c.score)	b.name
3.1765714285714293	"Nike"
3.2142857142857144	"Bosch"
3.2075471698113214	"Adidas"
3.0000000000000001	"Samsung"
1.5	"Apple"
2.75	"Ikea"
2.0	"Luminarc"

است:

کوئری چهارم:

برای اجرای این کوئری از رابطه‌ی commented استفاده کردیم تا امتیاز هر محصول را بدست بیاوریم. سپس میانگین امتیاز برای هر محصول را با with جدا کردیم. سپس در شرط where، با استفاده از regex رکوردهایی را برگرداندیم که در عنوان فارسی آن‌ها کلمه‌ی کفش وجود داشته باشد و میانگین امتیاز که در مرحله‌ی قبلی بدست آمد برای آن‌ها بیشتر از ۳ باشد. بخشی از نتیجه را می‌توانید در تصویر زیر مشاهده کنید.

neo4j\$ match (u:User)-[c:commented]-(p:Product) with avg(c.score) as avg_score, p.id as product_id, p.product_title as product_title where product_title =~ ".*کفشه.*" and avg_score > 3 return product_id, product_title, avg_score			
product_id	product_title	avg_score	
"728744"	کفشه دویین مرد اند آدیداس مدل "Cloudfoam Lite Racer"	4.0	
"796526"	کفشه مخصوص پیاده روی مرد اند آدیداس مدل "TERREX-GRAY 2018"	4.333333333333333	
"797531"	کفشه مخصوص دویین مرد اند آدیداس مدل "Energy Cloud"	3.8	
"758873"	کفشه ورزشی مخصوص دویین و پیاده روی مرد اند آدیداس مدل "Ultra Boost"	4.0	
"802096"	کفشه راحتی مرد اند آدیداس مدل "Topanga Clean"	5.0	
"751599"	کفشه مخصوص دویین و پیاده روی آدیداس مدل "ADIPRENE 2018"	5.0	
"728942"	کفشه مخصوص دویین زنانه "Swift"	4.666666666666667	
"787059"	کفشه مخصوص پیاده روی مرد اند شایگن مدل "Nike airmax bl.red"	4.5	
"797515"	کفشه مخصوص دویین مرد اند آدیداس مدل "Crazy Move"	4.0	
"776368"	کفشه ورزشی مخصوص دویین مرد اند شایگن مدل "Shield Zoom Winflo 4.5"	4.0	

کوئری پنجم:

برای مشخص کردن محصولات مدنظر، از دو رابطه‌ی commented و have استفاده شده است. چرا که ثبت نظر برای محصولات صورت می‌گیرد و برندها شامل نام برنده می‌شوند در نتیجه ما به join این اطلاعات احتیاج داریم. در بخش where رکوردهایی را جدا کردیم که نام آن‌ها brand است و امتیاز داده شده به محصول آن‌ها کمتر از ۳ است. و سپس با کلیدواژه‌ی delete رابطه‌های commented را از دیتابیس حذف کردیم. در نتیجه‌ی این دستور، ۸ رابطه از دیتابیس حذف شد.

neo4j\$ match (u:User)-[c:commented]-(p:Product)<-[h:have]-(b:Brand) where b.name = 'Nike' and c.score < 3 delete c			
Deleted 8 relationships, completed after 21 ms.			

کوئری ششم:

برای نوشتن کوئری این بخش، از سه رابطه‌ی buy، member و contain استفاده کردیم. از member استفاده می‌کنیم چرا که باید هر محصول را به کنگره‌ی آن وصل کنیم. سپس از contain استفاده می‌کنیم که رسیدهایی که در آن‌ها محصولات خریداری شده‌اند پیدا شوند و سپس با استفاده از buy می‌کنیم که رسیدهایی که در آن‌ها تمام اطلاعات لازم را داریم. با استفاده از این joinها یوزرهای مسئول ثبت آن رسیدها را بدست می‌آوریم. در قسمت return، با استفاده از دستور case به ازای هر رنج سنی، یک رشته نمایانگر گروه سنی خواسته شده برمی‌گردانیم و تعداد خریدهای هر دسته در هر کنگره‌ی را به تفکیک گروه سنی برمی‌گردانیم.

برای نوشتن نتیجه‌ای این کوئری در فایل csv از فرمت apoc برای نوشتن در فایل استفاده می‌کنیم که در تصویر زیر این فرمت هم قابل مشاهده است.

file	source	format	nodes	relationships	properties	time	rows	batchSize	batches	done	data
"pb_q6.csv"	"statement: col(3)"	"csv"	0	0	87	16	29	20000	1	true	null

کوئری هفتم:

برای این کوئری هم از سه رابطه‌ی `buy`, `have` و `contain` استفاده کردیم. با استفاده از رابطه‌ی `have` هر محصول را برندهش مربوط کردیم. سپس با استفاده از رابطه‌ی `contain` رسیدهایی که محصولات در آن‌ها حضور داشته‌اند را بدست آوردیم تا نهایتاً با رابطه‌ی `buy` بتوانیم به خریدار آن محصولات برسیم. تعداد رسیدهای مربوط به هر نام برنده، تعداد مشتریان آن‌ها هم هست. نهایتاً برای نوشتن در فایل `csv`، مشابه قسمت قبل عمل کردیم.

```
with "
    match (b:Brand)-[h:have]-(p:Product)->[c:contain]-(r:Receipt)->[x:buy]-(u:User) return b.name, count(x) as
customer_count order by customer_count desc;
" as query
CALL apoc.export.csv.query(query, "pb_q7.csv", {})
YIELD file, source, format, nodes, relationships, properties, time, rows, batchSize, batches, done, data;
RETURN file, source, format, nodes, relationships, properties, time, rows, batchSize, batches, done, data;
```

"file"	"source"	"format"	"nodes"	"relationships"	"properties"	"time"	"rows"	"batchSize"	"batches"	"done"	"data"
"pb_q7.csv"	"statement: cols(2)"	"csv"	0	0	14	6	7	20000	1	true	null