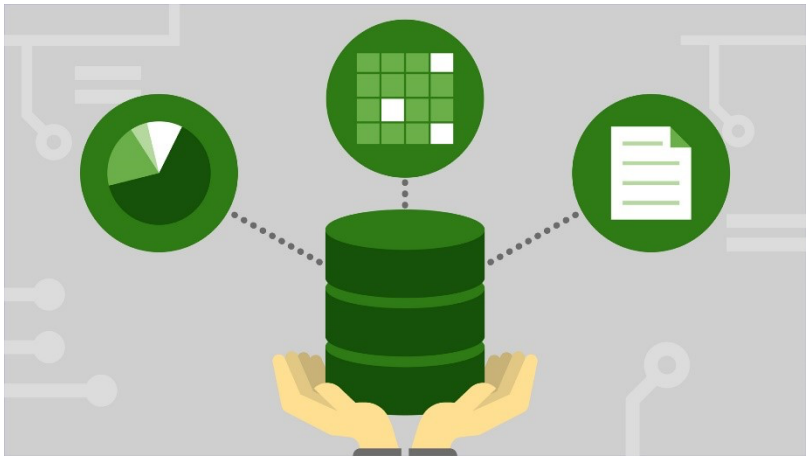
به نام خدا

دانشگاه تهران
پردیس دانشکده‌های فنی
دانشکده برق و کامپیوتر



آزمایشگاه پایگاه داده

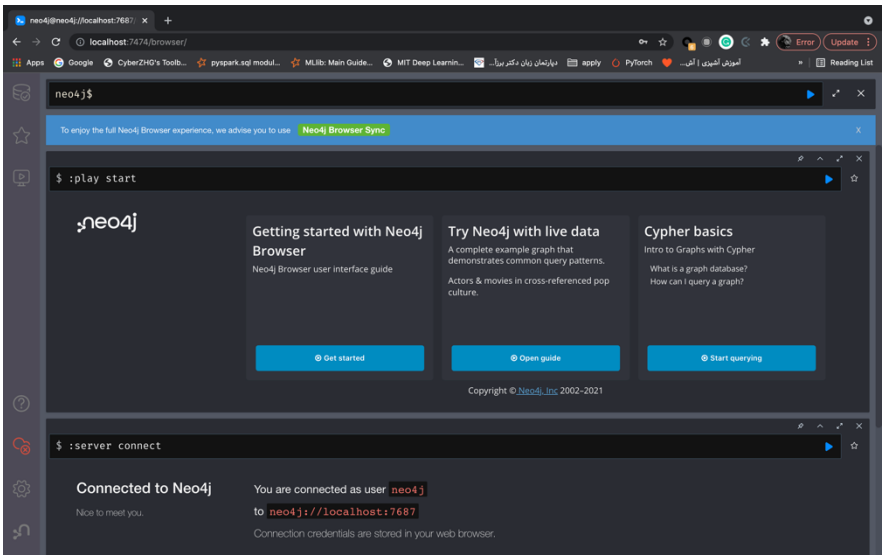
دستور کار شماره ۳

آتیه آرمین
۸۱۰۱۹۷۶۴۸

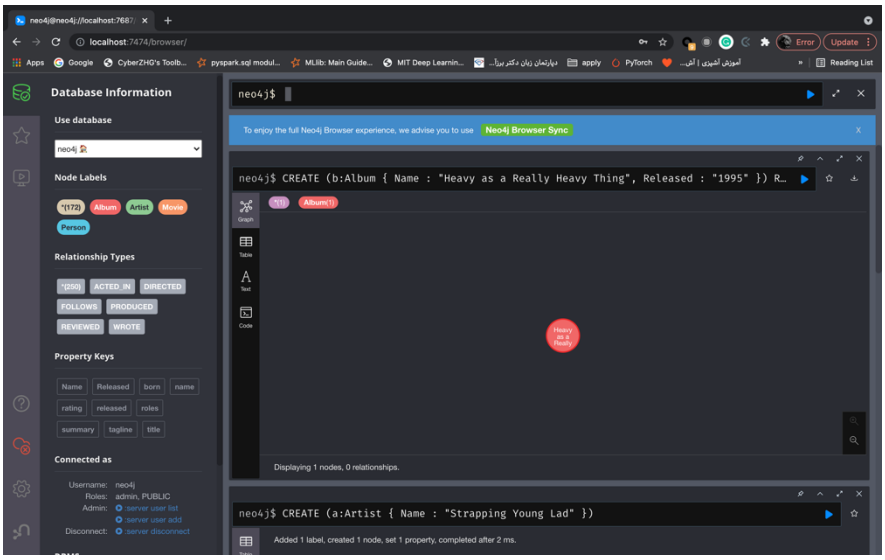
مهرماه ۱۴۰۰

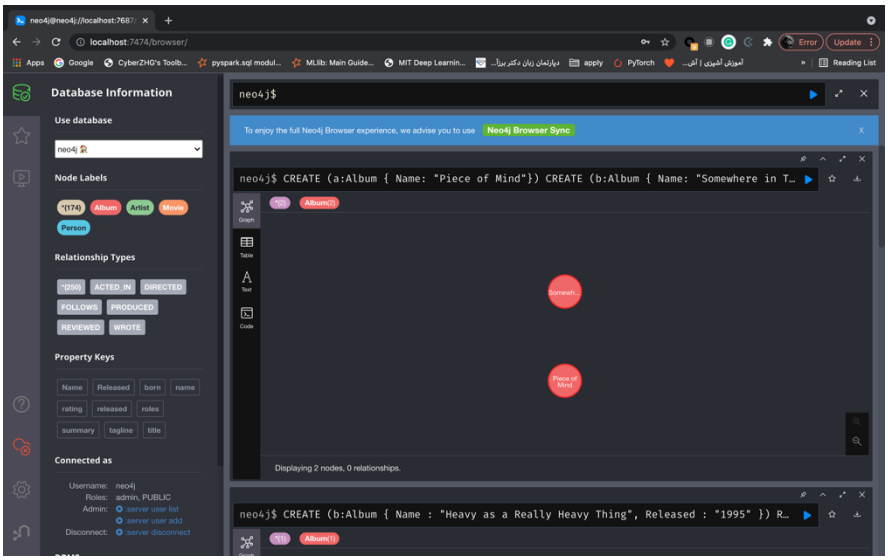
گزارش دستورکار انجام شده

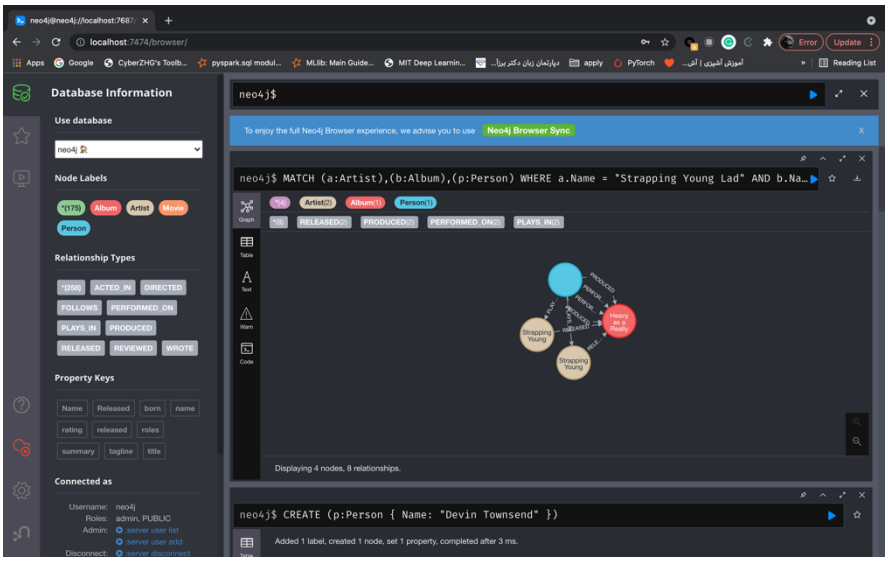
در بخش اول به آموزش ساده سایت Quackit پرداختم. خروجی دستورات مختلف اجرا شده را در ادامه مشاهده میفرمایید.
ابتدا پس از دانلود neo4j، URL ایجاد شده را اجرا کردم.

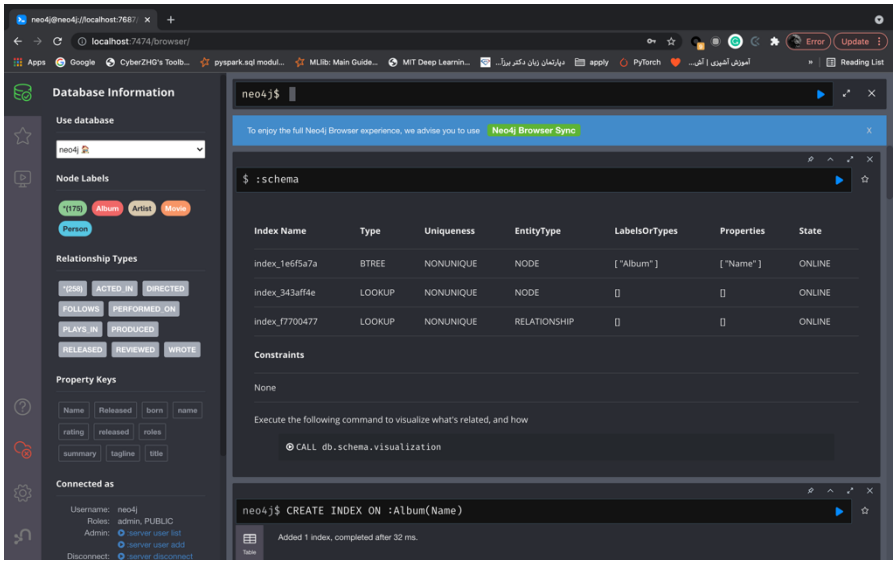


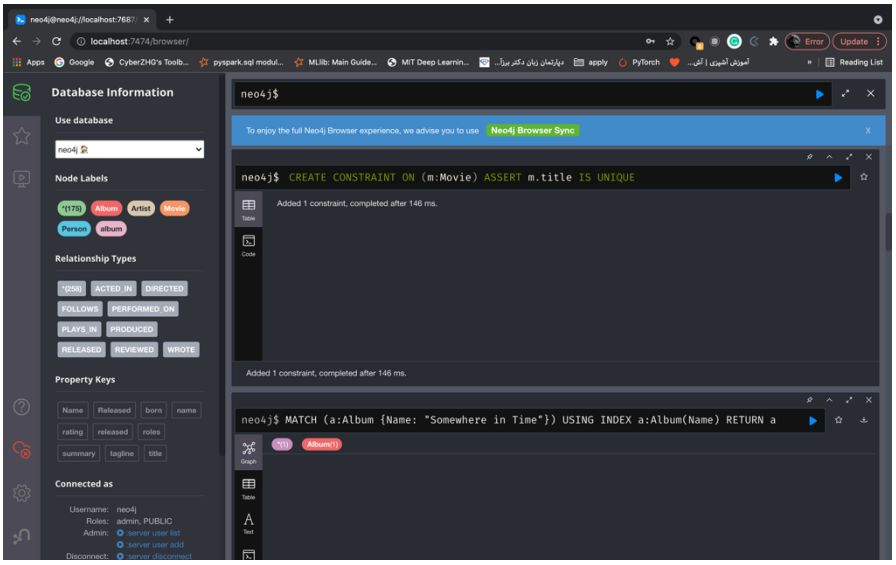
در ادامه خروجی دستورات مختلف سایفر را مشاهده میکنید.

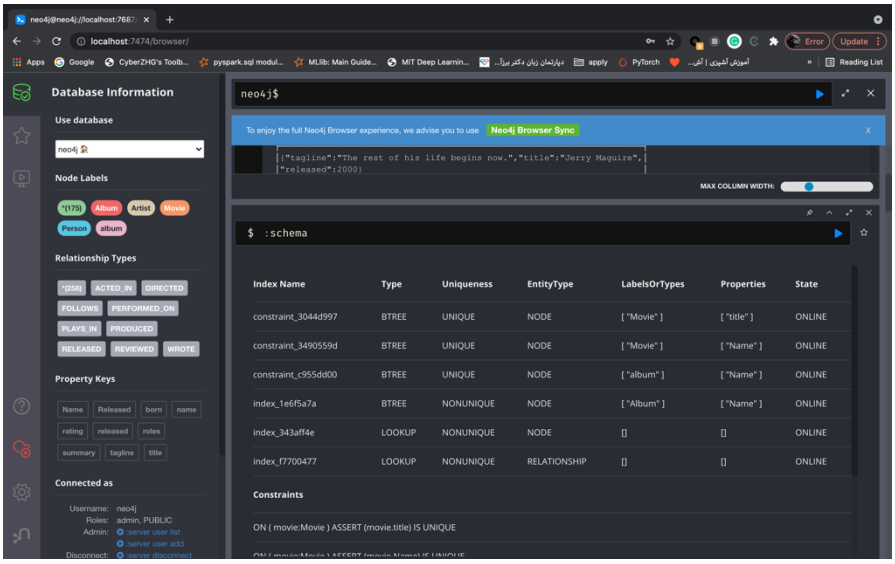


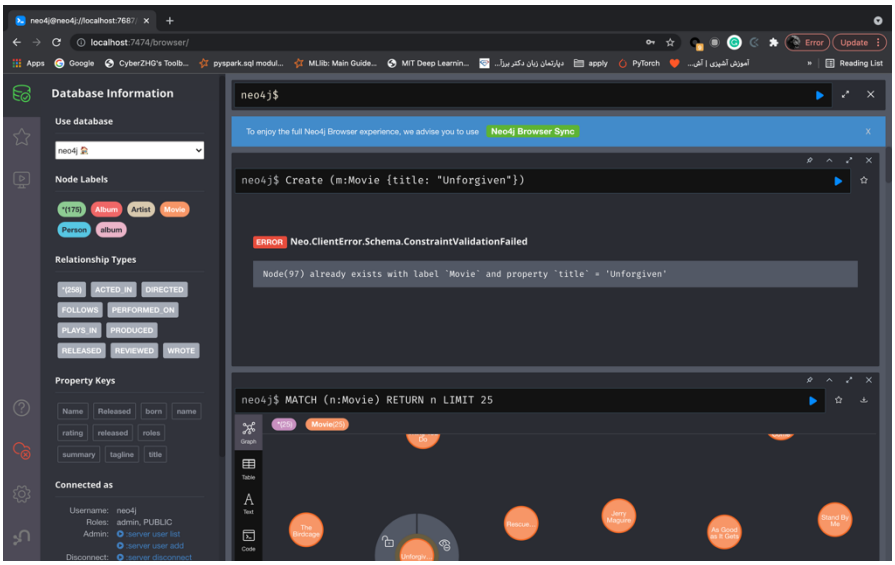

Database Information
Use database
neo4j
Node Labels
Relationship Types
Property Keys
Connected as
neo4j$
To enjoy the full Neo4j Browser experience, we advise you to use Neo4j Browser Sync
neo4j$ CREATE (a:Album { Name: "Piece of Mind"}) CREATE (b:Album { Name: "Somewhere in T…
Displaying 2 nodes, 0 relationships.
neo4j$ CREATE (b:Album { Name : "Heavy as a Really Heavy Thing", Released : "1995" }) R…

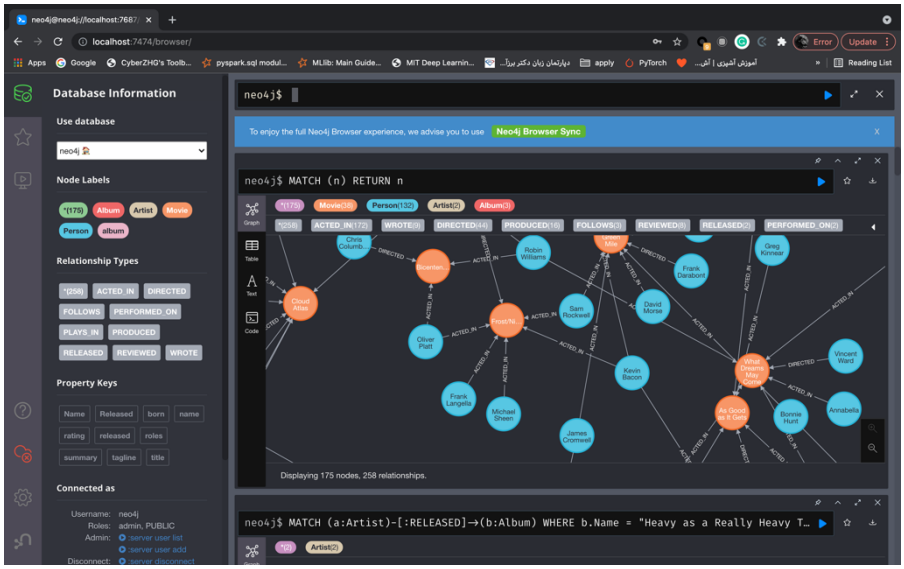

Database Information
Use database
neo4j
Node Labels
Relationship Types
Property Keys
Connected as
neo4j$
To enjoy the full Neo4j Browser experience, we advise you to use Neo4j Browser Sync
neo4j$ MATCH (a:Artist),(b:Album),(p:Person) WHERE a.Name = "Strapping Young Lad" AND b.Na…
Displaying 4 nodes, 8 relationships.
neo4j$ CREATE (p:Person { Name: "Devin Townsend" })
Added 1 label, created 1 node, set 1 property, completed after 3 ms.

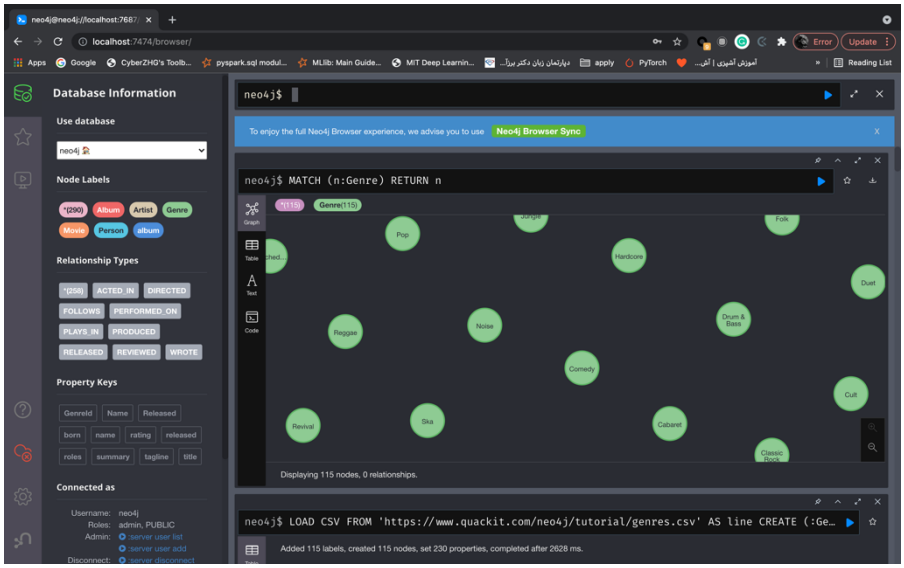

Database Information
Use database
neo4j
Node Labels
Relationship Types
Property Keys
Connected as
neo4j$
To enjoy the full Neo4j Browser experience, we advise you to use Neo4j Browser Sync
$:schema
Index Name
Type
Uniqueness
EntityType
LabelsOrTypes
Properties
State
index_1e6f5a7a
BTREE
NONUNIQUE
NODE
["Album"]
["Name"]
ONLINE
index_343aff4e
LOOKUP
NONUNIQUE
NODE
[]
[]
ONLINE
index_f7700477
LOOKUP
NONUNIQUE
RELATIONSHIP
[]
[]
ONLINE
Constraints
None
Execute the following command to visualize what's related, and how
CALL db.schema.visualization
neo4j$ CREATE INDEX ON :Album(Name)
Added 1 index, completed after 32 ms.

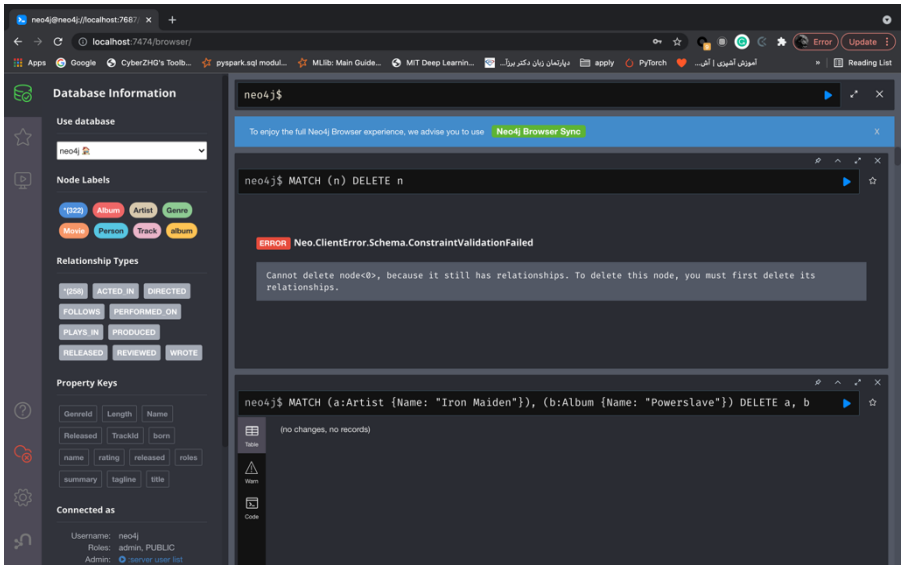

Database Information
Use database
neo4j
Node Labels
Relationship Types
Property Keys
Connected as
neo4j$
To enjoy the full Neo4j Browser experience, we advise you to use Neo4j Browser Sync
neo4j$ CREATE CONSTRAINT ON (m:Movie) ASSERT m.title IS UNIQUE
Added 1 constraint, completed after 146 ms.
neo4j$ MATCH (a:Album {Name: "Somewhere in Time"}) USING INDEX a:Album(Name) RETURN a

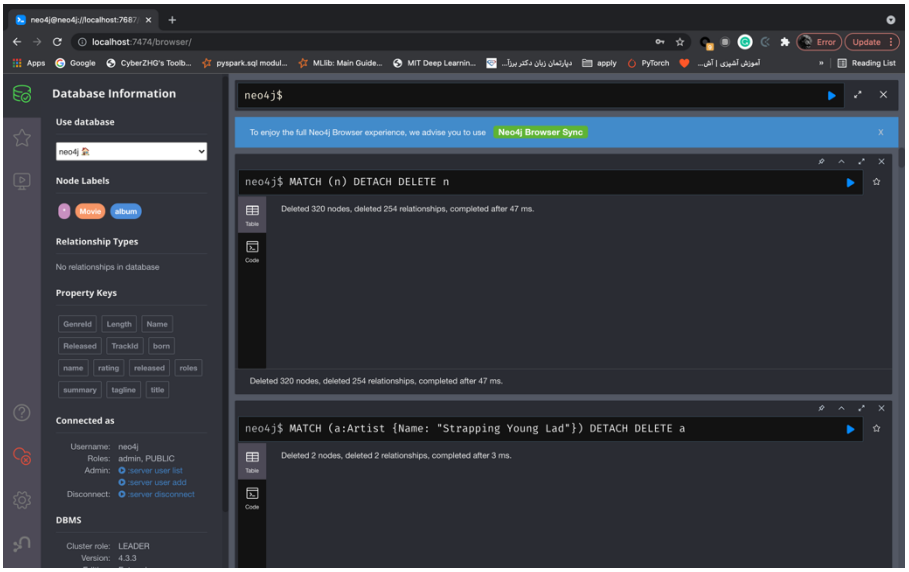

Database Information
neo4j$
To enjoy the full Neo4j Browser experience, we advise you to use Neo4j Browser Sync
$:schema
Index Name
Type
Uniqueness
EntityType
LabelsOrTypes
Properties
State
constraint_3044d997
constraint_3490559d
constraint_c955dd00
index_1e6f5a7a
index_343aff4e
index_f7700477
Constraints
ON (movie:Movie) ASSERT (movie.title) IS UNIQUE


Database Information
neo4j$
To enjoy the full Neo4j Browser experience, we advise you to use Neo4j Browser Sync
neo4j$ Create (m:Movie {title: "Unforgiven"})
ERROR Neo.ClientError.Schema.ConstraintValidationFailed
Node(97) already exists with label `Movie` and property `title` = 'Unforgiven'
neo4j$ MATCH (n:Movie) RETURN n LIMIT 25


Database Information
Use database
neo4j
Node Labels
Relationship Types
Property Keys
Connected as
neo4j$
To enjoy the full Neo4j Browser experience, we advise you to use Neo4j Browser Sync
neo4j$ MATCH (n) RETURN n
Displaying 175 nodes, 258 relationships.
neo4j$ MATCH (a:Artist)-[:RELEASED]→(b:Album) WHERE b.Name = "Heavy as a Really Heavy T…

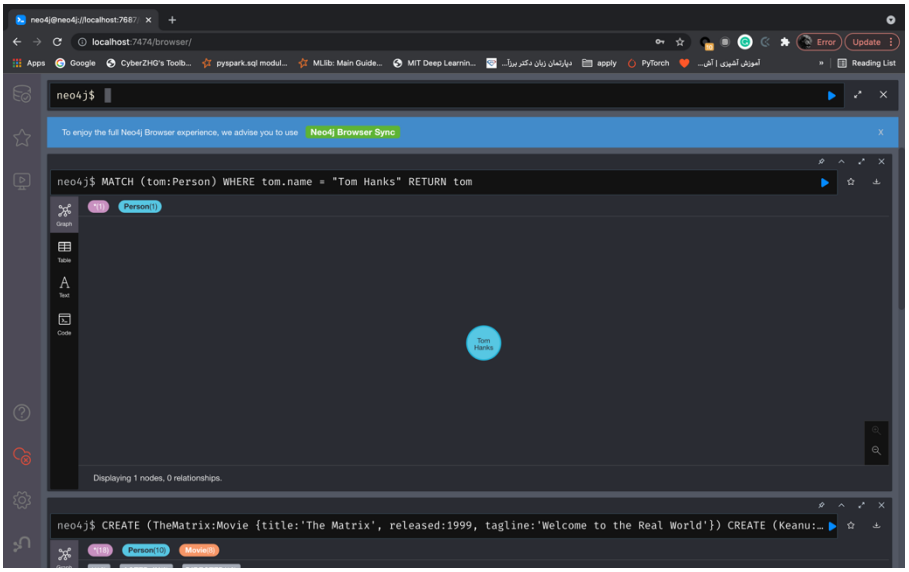

Database Information
Use database
neo4j
Node Labels
Relationship Types
Property Keys
Connected as
neo4j$
To enjoy the full Neo4j Browser experience, we advise you to use Neo4j Browser Sync
neo4j$ MATCH (n:Genre) RETURN n
Displaying 115 nodes, 0 relationships.
neo4j$ LOAD CSV FROM 'https://www.quackit.com/neo4j/tutorial/genres.csv' AS line CREATE (:Ge…
Added 115 labels, created 115 nodes, set 230 properties, completed after 2628 ms.


Database Information
Use database
neo4j
Node Labels
Relationship Types
Property Keys
Connected as
neo4j$
To enjoy the full Neo4j Browser experience, we advise you to use Neo4j Browser Sync
neo4j$ MATCH (n) DELETE n
ERROR Neo.ClientError.Schema.ConstraintValidationFailed
Cannot delete node<0>, because it still has relationships. To delete this node, you must first delete its relationships.
neo4j$ MATCH (a:Artist {Name: "Iron Maiden"}), (b:Album {Name: "Powerslave"}) DELETE a, b
(no changes, no records)

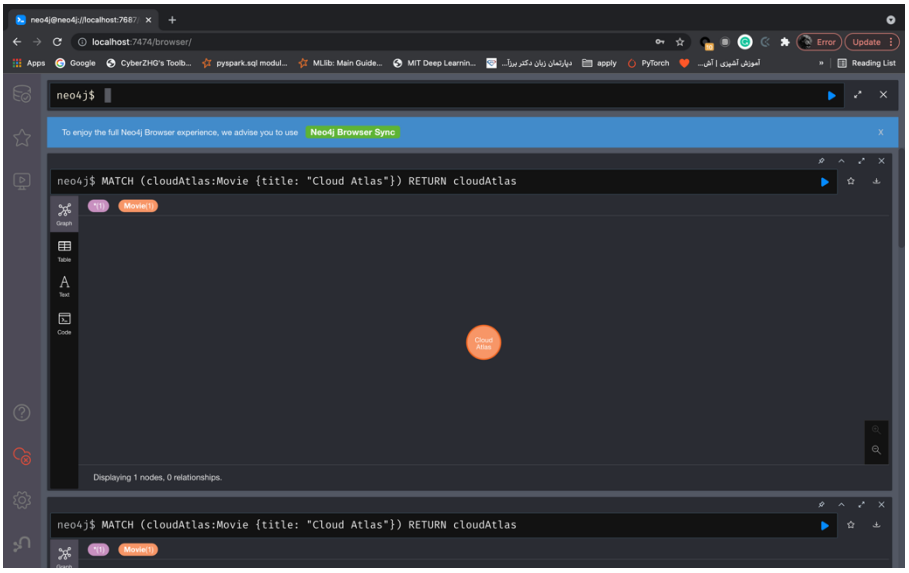


در بخش دوم به یادگیری سایفر به کمک مستندات رسمی ساید noe4j پرداختم.

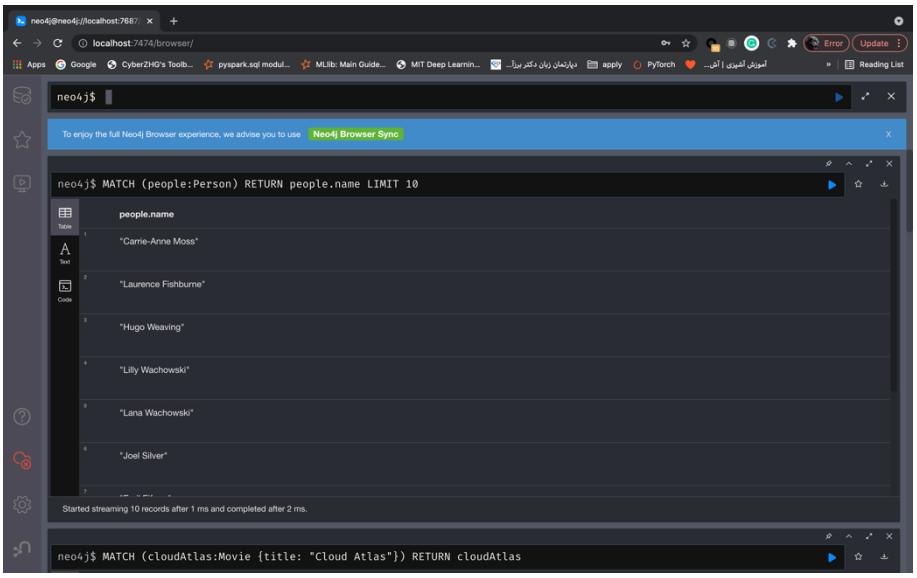
در دستور زیر میخواهیم یک بازیگر با نامی مشخص را بیابیم که با دستور MATCH / WHERE میتوانیم به خواسته‌مان برسیم.



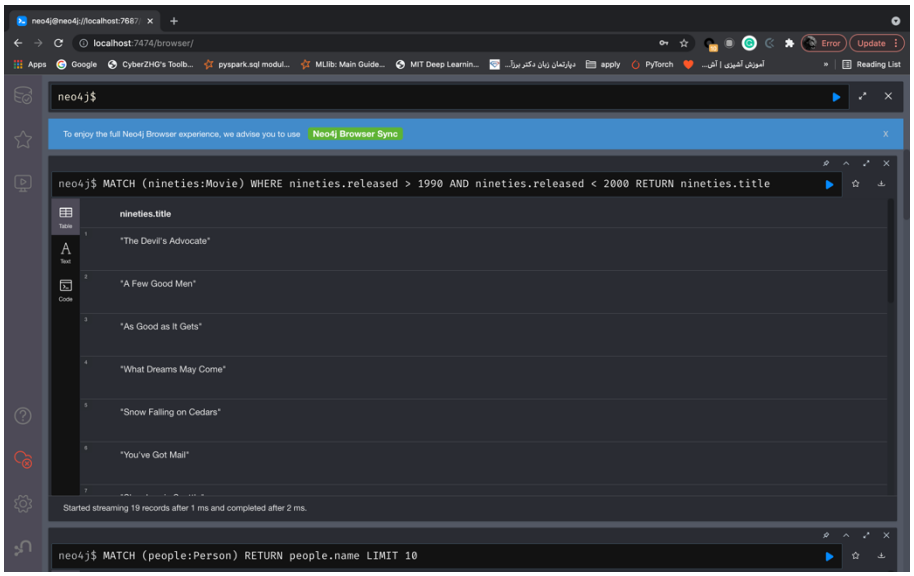
در دستور زیر میخواهیم فیلمی با یک نام مشخص را بیابیم. برای اینکار از روش دیگری استفاده میکنیم و به جای where از syntax دیگری استفاده میکنیم.



در دستور زیر میخواهیم نام ۱۰ نفر در گراف را پیدا کنیم. برای اینکار نام تمام افراد را پیدا میکنیم ولی تنها ۱۰ تای آن ها را نمایش میدهیم.

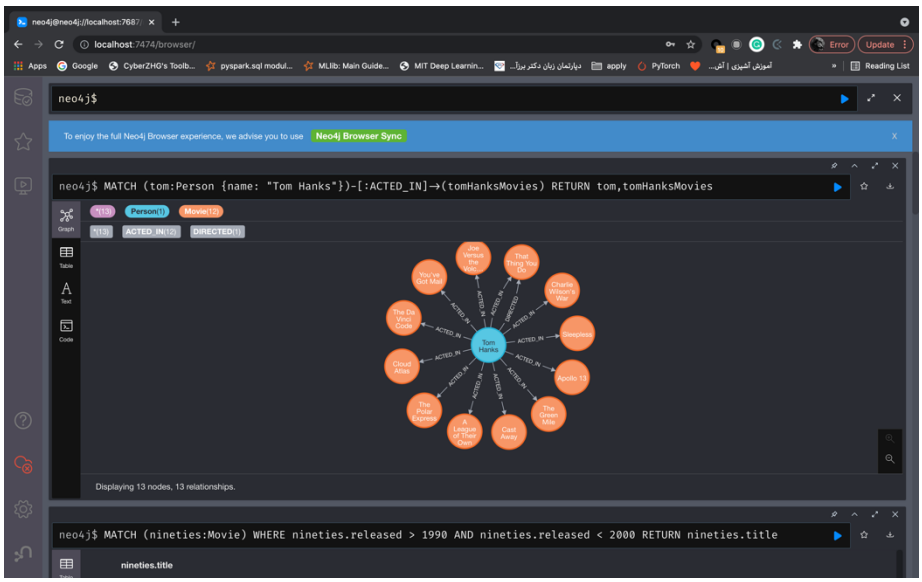


در دستور زیر تمام فیلم هایی که در سال ۱۹۹۰ اکران شده اند را میخواهیم بیابیم که برای این کار دوباره از match و where استفاده میکنیم.

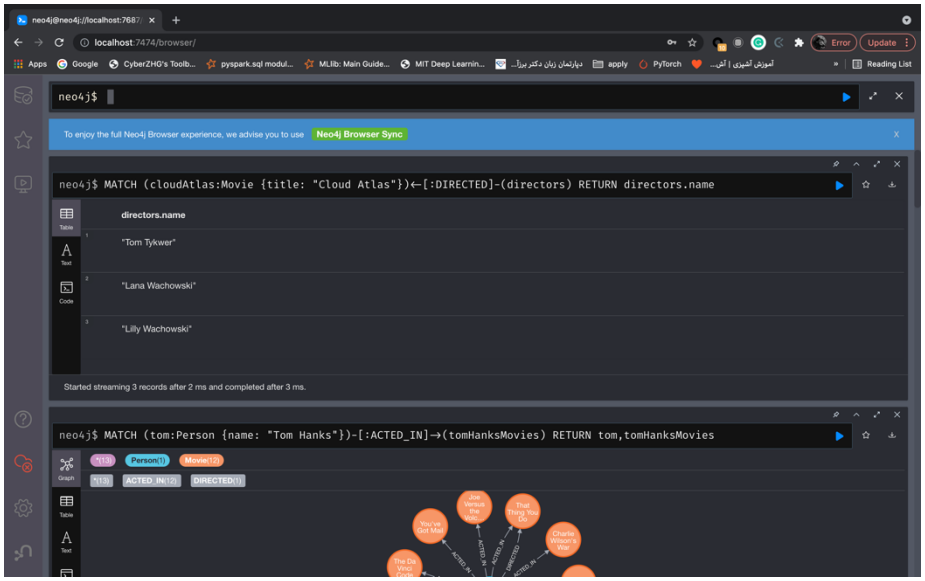


در دستورهای زیر تلاش به پیدا کردن یک الگو در گرافمان داریم.

در دستور زیر میخواهیم تمام فیلم هایی که tom hanks در آن ها است را بیابیم. برای این کار از ACTED_IN relationship استفاده میکنیم و راس هایی را می‌یابیم که با این رابطه با راس tom hanks هستند.

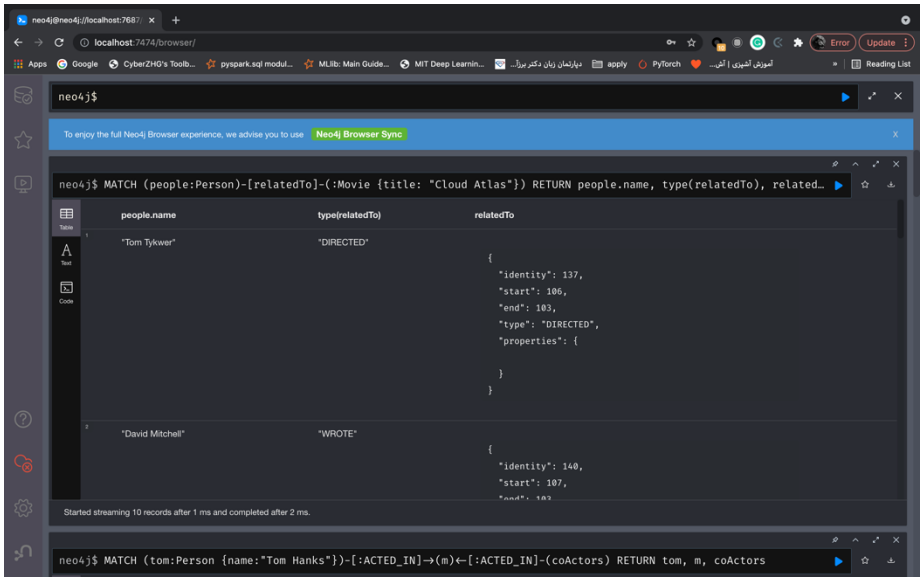


در دستور زیر به دنبال افرادی هستیم که کارگردانی فیلم cloudAtlas را کرده اند. برای این کار هم مانند دستور قبلی تمام راس هایی که با رابطه‌ی DIRECTED به فیلم cloud Atlas متصل هستند را می‌یابیم.



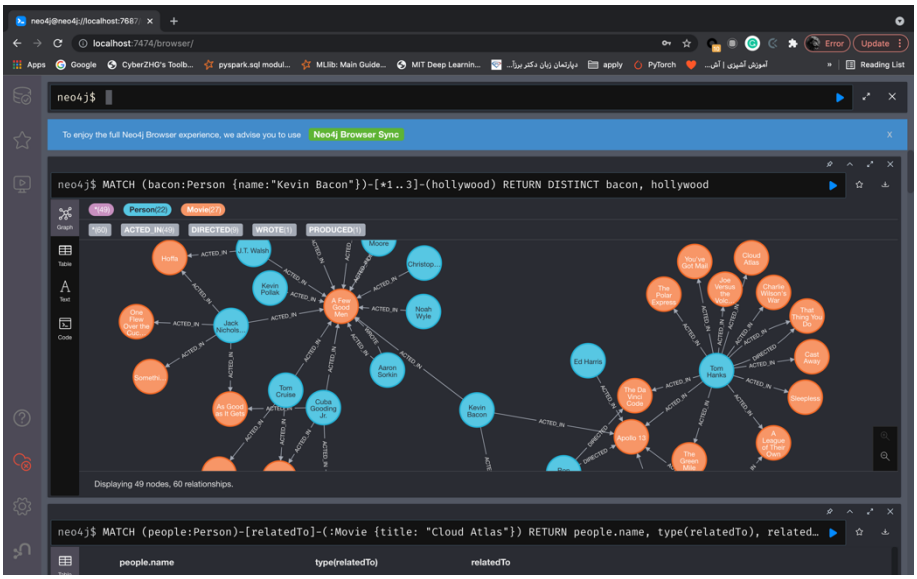
در دستور زیر به دنبال افرادی هستیم که همراه با tom hanks در یک فیلم بازی کردند. برای این کار ابتدا فیلم هایی که tom hanks در آن ها بازی کرده و سپس افراد دیگری که با رابطه‌ی ACTED_IN به آن ها متصل شده اند را میبابیم.

در دستور زیر میخواهیم اطلاعات درباره ارتباط های افراد با فیلم cloud atlas را دریافت کنیم. برای اینکار تمام افرادی که به نحوی با این فیلم در ارتباط هستند را با relatedTo می یابیم و تایپ این ارتباط را نمایش میدهیم.

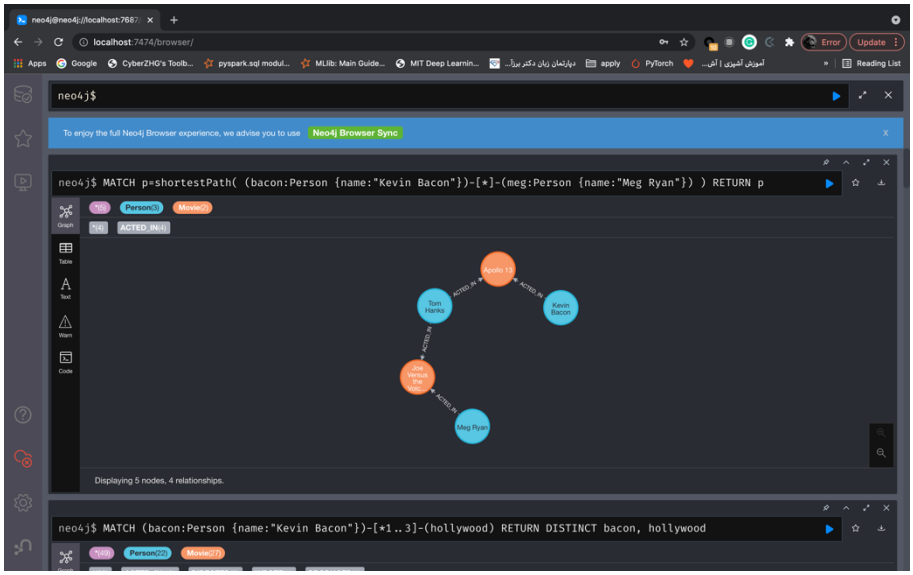


در دستورات زیر میخواهیم به چند سوال درباره گرافمان پاسخ دهیم.

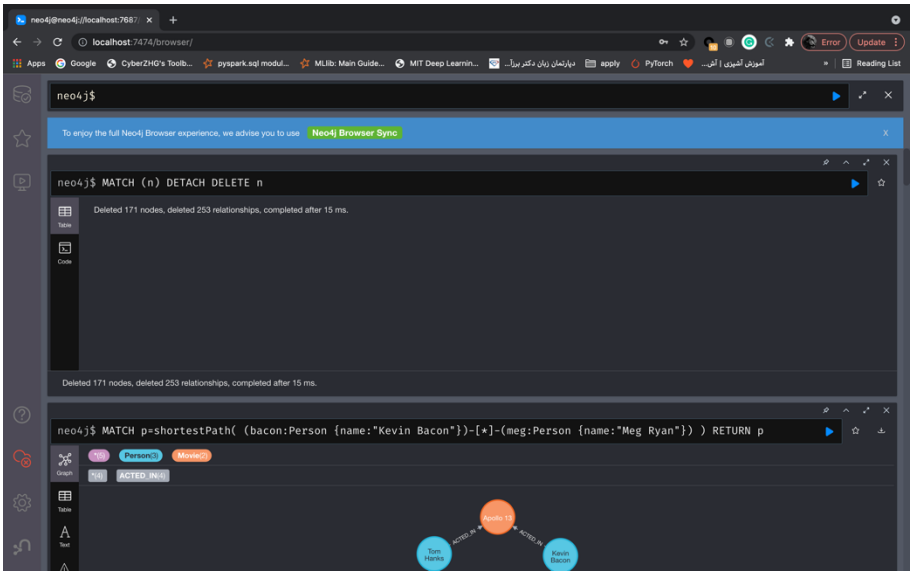
در دستور زیر به دنبال تمام فیلم ها یا افرادی هستیم که حداکثر ۳ hop از kevin bacon در گراف فاصله دارند. برای این کار از syntax ای به صورت [*1..3] استفاده میکنیم.

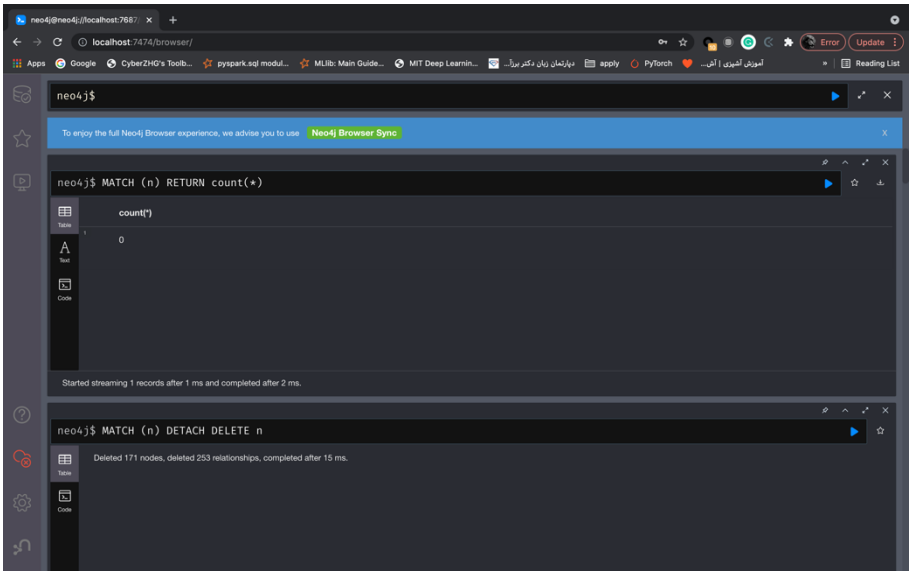


در دستور زیر میخواهیم کوتاه ترین مسیر بین kevin bacon و meg ryan را در گرافمان پیدا کنیم. برای این کار از syntaxای به صورت shortestPath() استفاده میکنیم و بین تمام مسیر های بین آن دو، کوتاه ترین را انتخاب میکنیم.



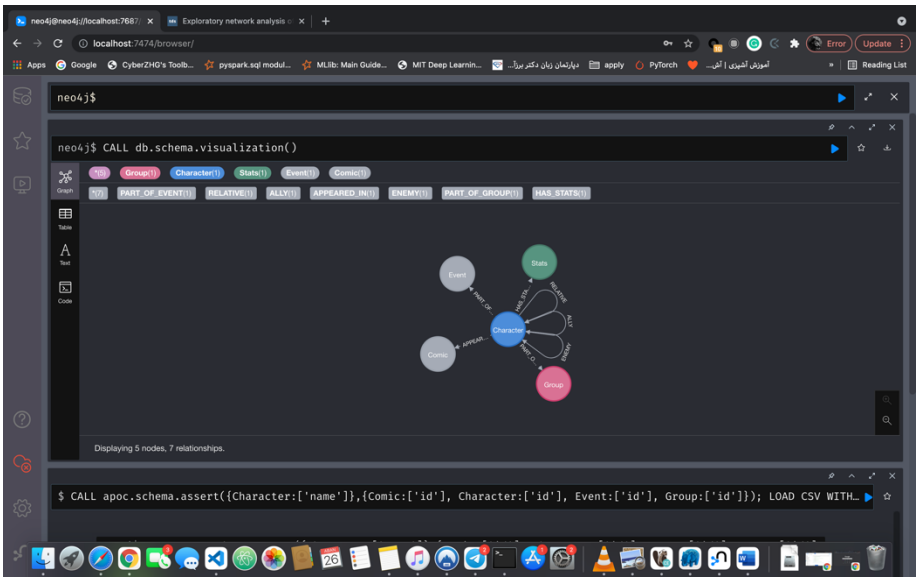
در دستورات بعدی به پاک کردن دیتاست میپردازیم. توجه داریم که راس ها تا وقتی که ارتباطات پاک نشوند، پاک نخواهند شد.



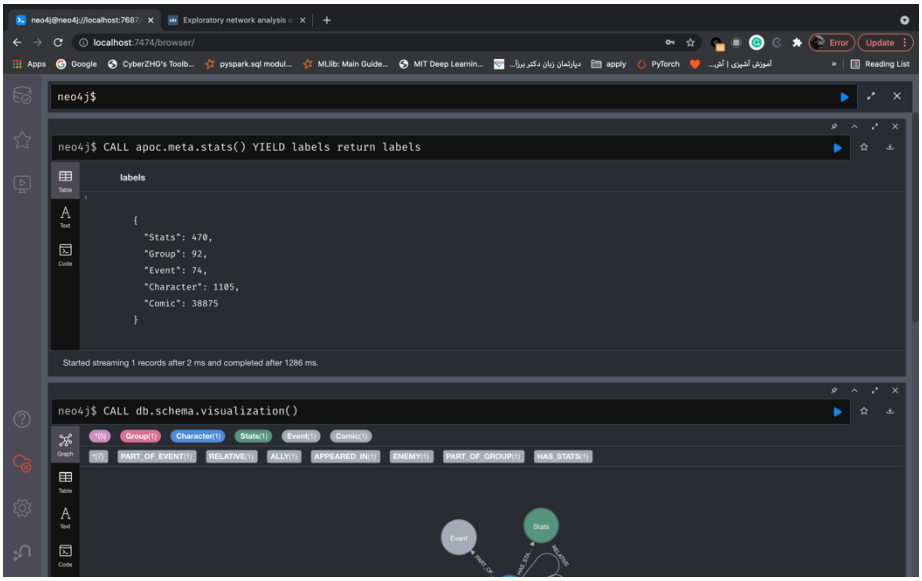


در ادامه به بررسی مقاله ذکر شده در صورت پروژه پرداختم. این مقاله سعی دارد گراف دیتاست دنیای marvel را آنالیز و بررسی کند. و در آخر با اعمال چند الگوریتم، دسته بندی خاصی را روی این گراف اعمال کند.

ابتدا همانطورکه در عکس زیر مشاهده میشود، داده ها را دانلود و شمای دیتا را نشان میدهیم.

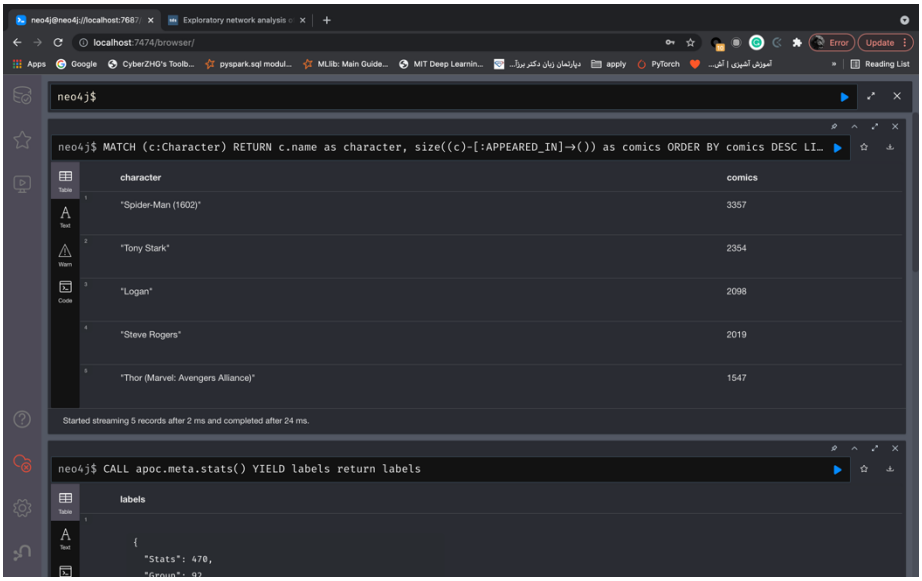


سپس برای آنکه کمی با اندازه گراف و بزرگی آن آشنا شویم از apoc.meta.stats() استفاده میکنیم.

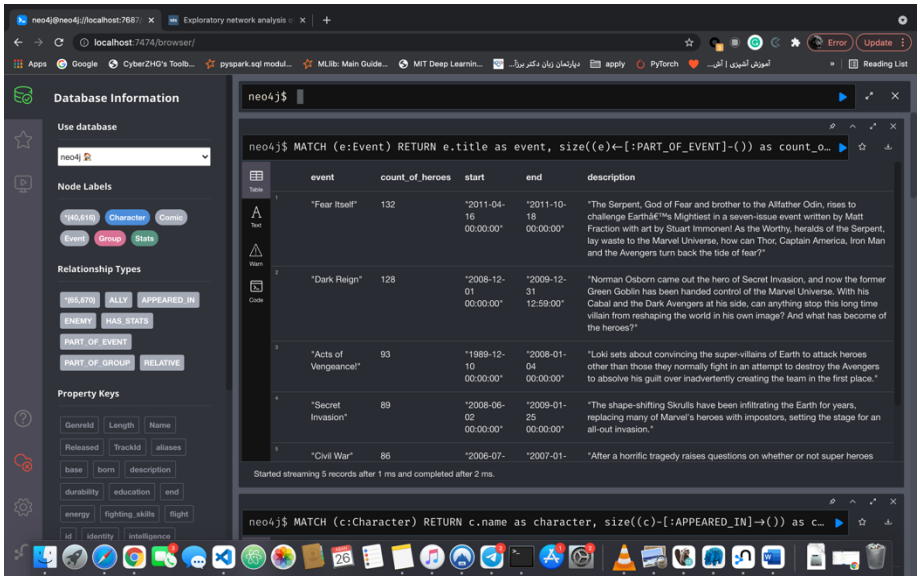


در ادامه به آنالیز گراف میپردازیم.

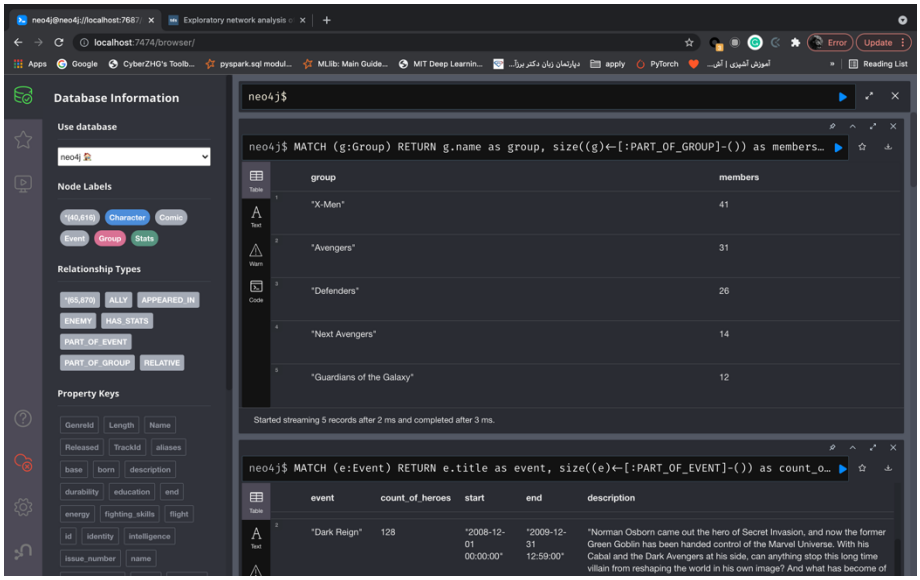
ابتدا به پیدا کردن کاراکتر هایی می پردازیم که بیشترین میزان تکرار را در comic ها دارند. برای اینکار جواب ها را به ۵ تا محدود میکنیم و ۵ کاراکتر پر تکرار را نمایش میدهیم.



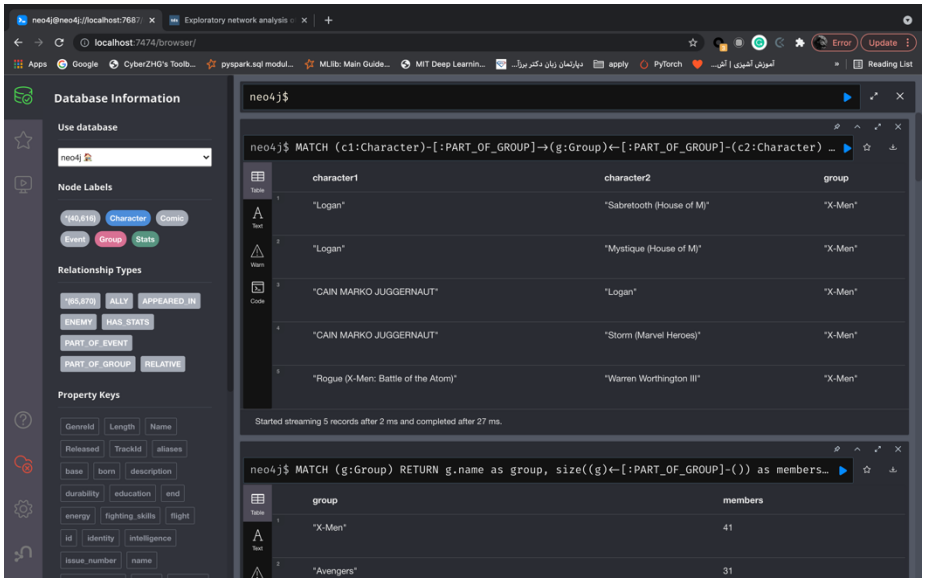
حال به بررسی event هایی میپردازیم که بیشترین hero ها در آن ها شرکت کرده اند و زمان شروع و پایان و همچنین توضیحات آن event ها را نمایش میدهیم. در این قسمت هم مانند قسمت قبل، نتایج را ۵ تا کاهش میدهیم.



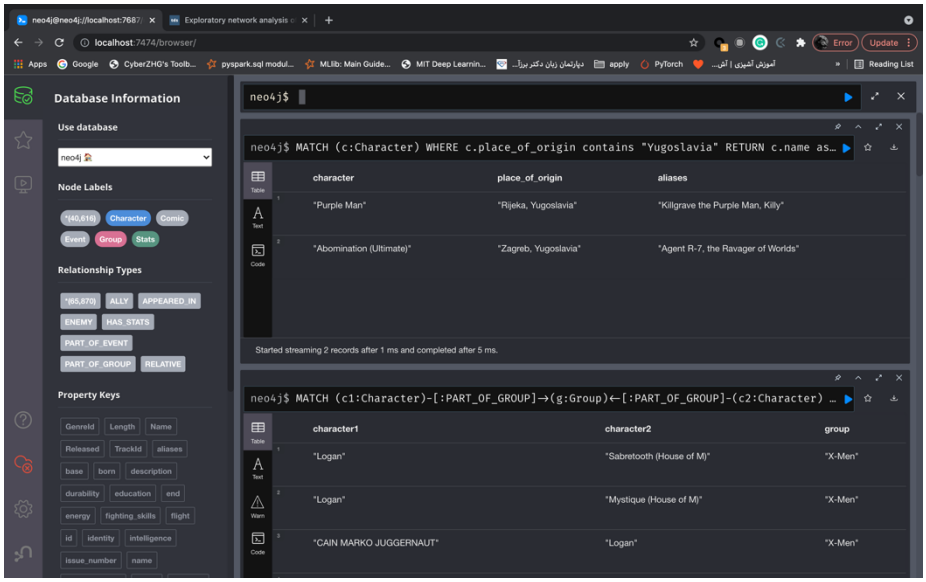
حال تعداد کاراکترهایی که عضو یک گروه از کاراکتر ها هستند را به دست می‌آوریم و ۵ تا از بزرگترین گروه ها را نمایش میدهیم.



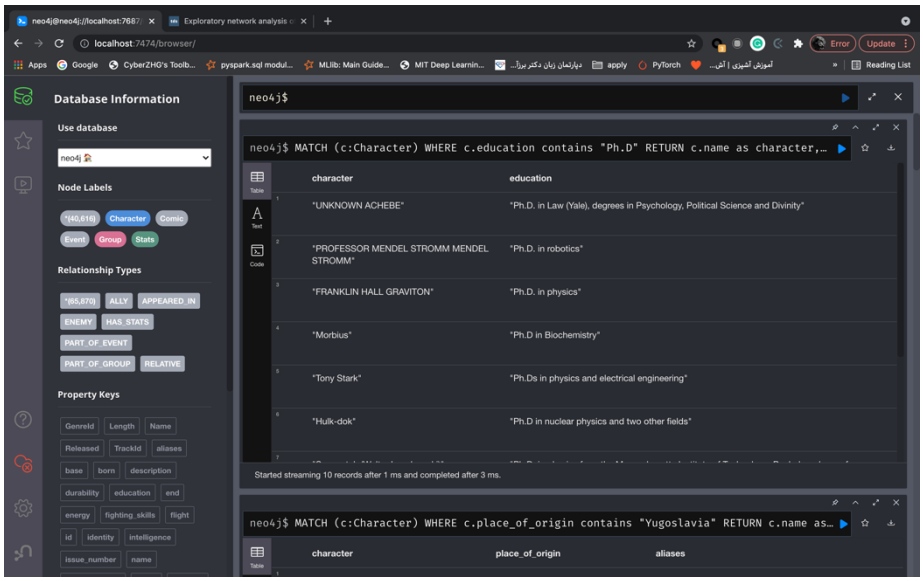
حال بررسی میکنیم که آیا افرادی در یک گروه هستند که باهم دشمن باشند؟ این کار را با بررسی رابطه‌ی enemyبین دو کاراکتری که هردو با یک گروه رابطه‌ی part of group دارند، انجام میدهیم.



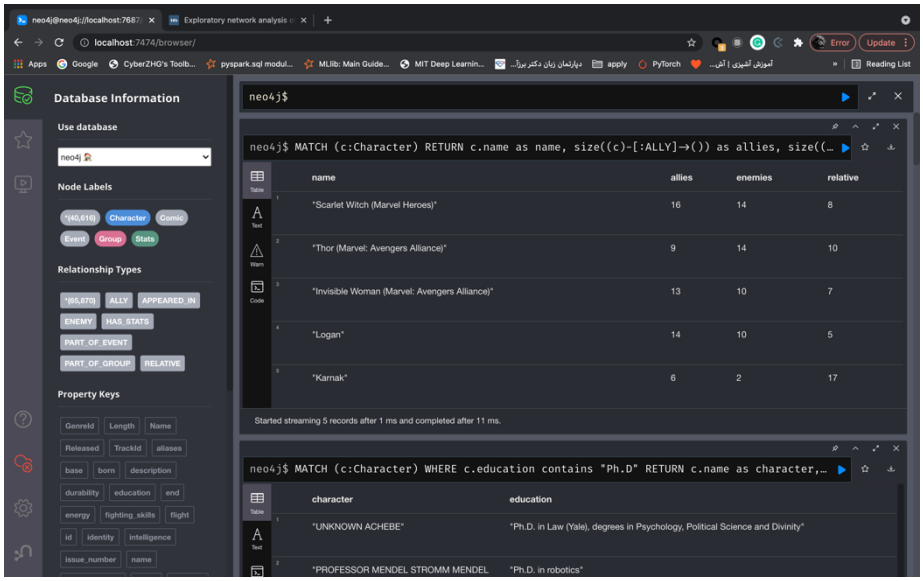
حال میخواهیم ببینیم چه کاراکتر هایی از Yugoslavia ظهور کرده اند. برای اینکار از عملگر contains استفاده کرده و کاراکتر هایی که place of origin شان شامل yugoslaviaباشد را جدا میکنیم.



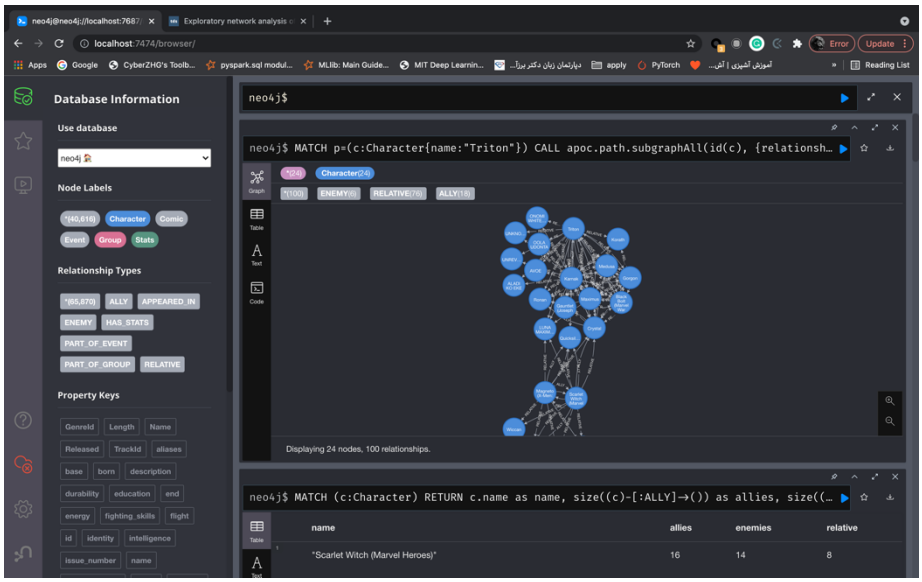
برای آنکه کمی بیشتر با کاراکتر ها آشنا شویم،به دنبال کاراکتر هایی میگردیم که مدرک Ph.D شان را گرفته اند و ۱۰ نفر آن ها را نمایش میدهیم.



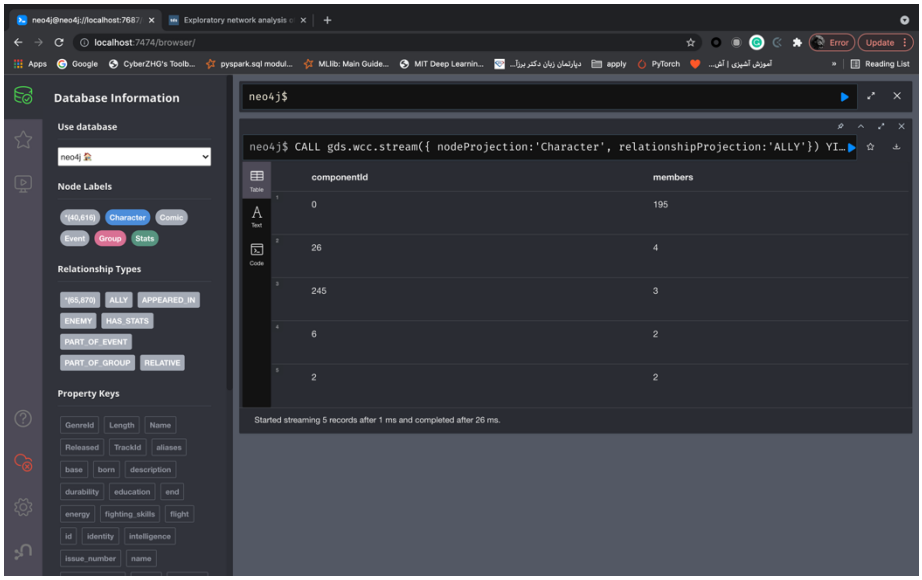
در ادامه به بررسی و آنالیز جامعه ها و community های بستگان و متحدان میپردازیم. برای اینکار ابتدا برای هر کاراکتر، تعداد بستگان، متحدان و دشمنان آن را پیدا میکنیم . این کار با بررسی روابط ENEMY, ALLY, RELATIVE انجام می شود.



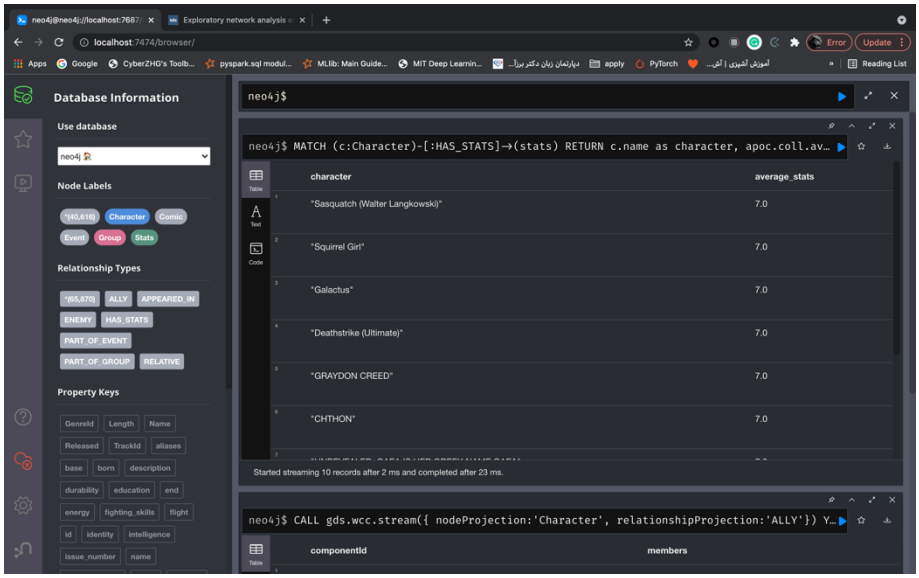
برای آنکه جامعه بستگان یک کاراکتر را بررسی کنیم میتوانیم از apoc.path.subgraphAll() استفاده کنیم و گراف آن را بیابیم.



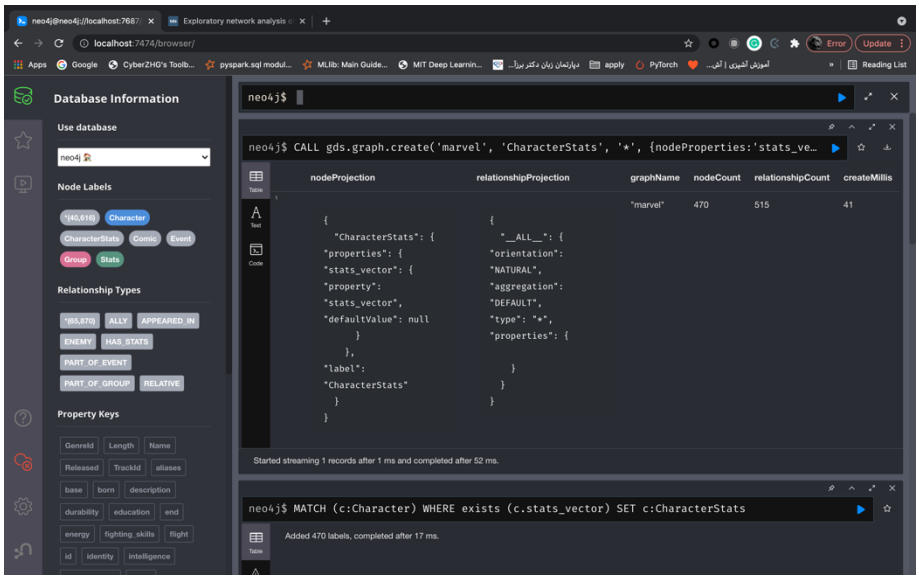
برای آنکه بتوانیم component هایی که به هم متصل نیستند را در یک شبکه بیابیم به دنبال weakly connected component ها میگردیم. در اینجا از weakly connected component ها استفاده میکنیم تا بزرگترین component از کاراکتر های متحد را بیابیم و ۵ تای آن ها را نمایش میدهیم.



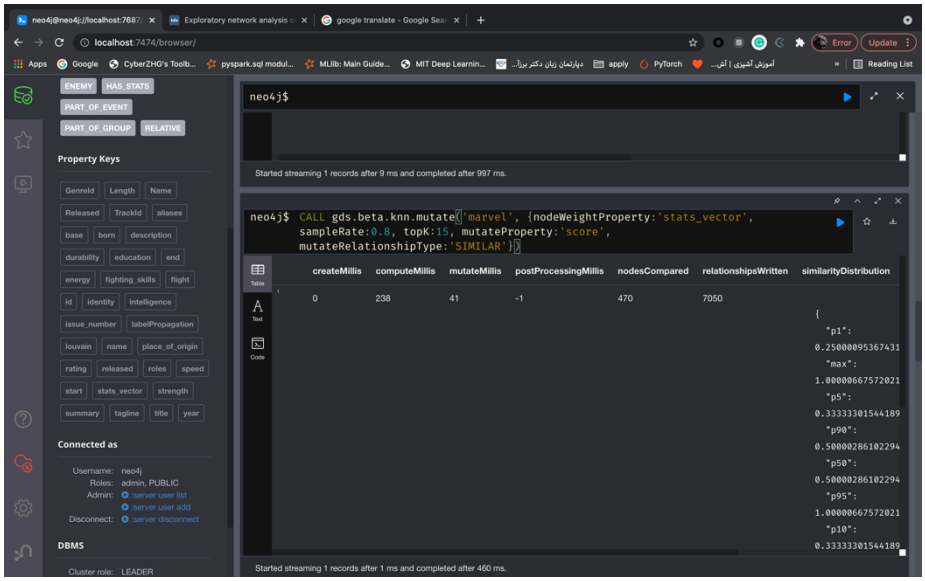
در ادامه به بررسی و آنالیز stat کاراکتر ها یا همان آمار skill های هر کاراکتر میپردازیم.
ابتدا از apoc.map.values() استفاده میکنیم که به ما در fetch کردن تمام خصوصیات یک گره بدون نوشتن صریح کلید های ویژگی کمک میکند.



حال به انجام ۳ الگوریتم متفاوت میپردازیم. ابتدا الگوریتم knn را اعمال میکنیم که در آن کاراکتر هایی که stat های آنها بیشتر به هم شبیه است و نزدیکتر است را در یک دسته بندی قرار میدهد.
در این مثال ابتدا یک vector میسازیم که این vector قرار است معیار ما قرار گیرد و بین هر دو قهرمان این vector مقایسه میشود. علاوه بر ویژگی هایی که در دیتاست هست، یک ویژگی دیگر که نشان دهنده این است که کاراکتر میتواند پرواز کند یا نه را هم به این vector اضافه میکنیم. حال گراف حاصل را میسازیم که همه node ها را با لیبل characterStats و stat_vector شان نشان میدهد.

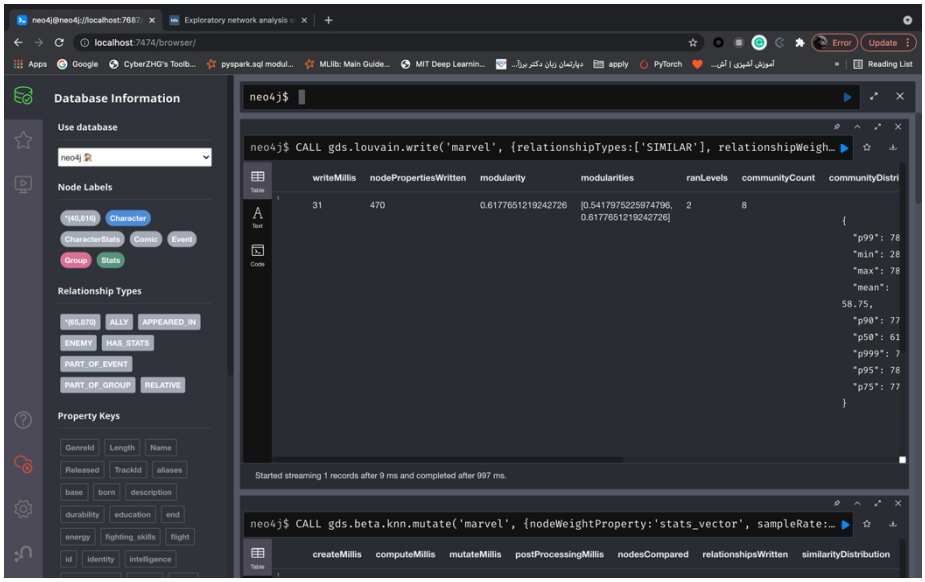


حال الگوریتم knn را با topK ۱۵ و sampleRate 0.8 اعمال میکنیم. در اینجا topK تعداد همسایه هایی است که برای هر راس پیدا میکند و sampleRate برای محدود کردن تعداد comparisons در هر راس است.

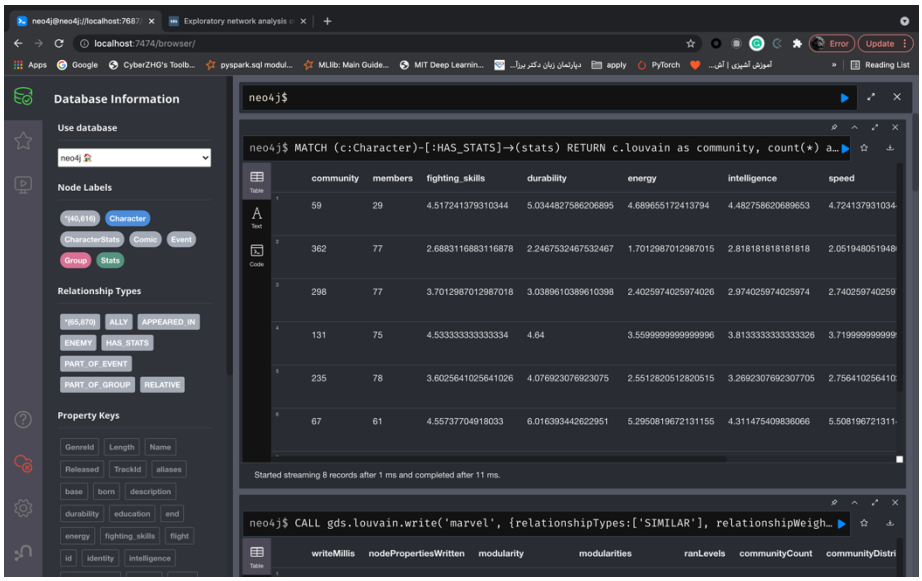


در ادامه به الگوریتم Louvain Modularity میپردازیم که هر دسته‌ای که knn ایجاد کرده است را مورد بررسی قرار میدهد و ساختار هر community را مورد بررسی قرار میدهد.

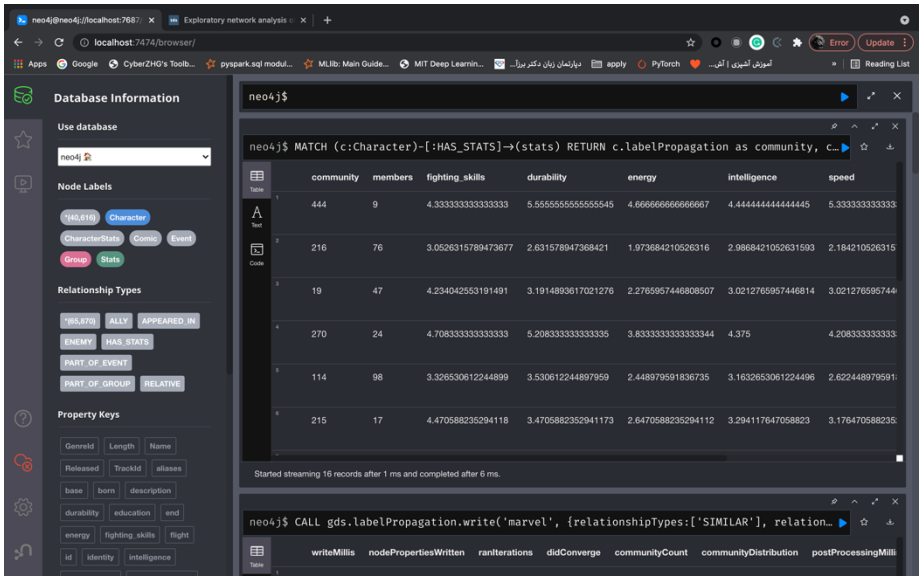
ابتدا با relationshipWeightProperty به الگوریتم میفهمانیم که هنگام بررسی ساختار باید وزن رابطه ها را در نظر بگیرد.



حال در هر community متوسط هر skill را محاسبه میکنیم.



کار بالا را با الگوریتم label propagation تکرار میکنیم و نتایج را باهم مقایسه میکنیم.



همانطور که مشخص است، label propagation تعداد بیشتری community پیدا میکند و برخی از آن ها تعداد کمتری دارند.