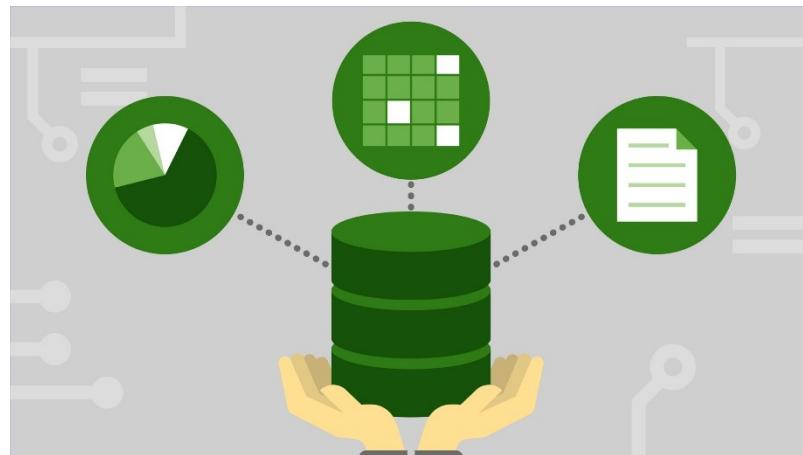


به نام خدا



دانشگاه تهران
پردیس دانشکده‌های فنی
دانشکده برق و کامپیوتر



آزمایشگاه پایگاه داده

دستور کار شماره ۳

آتبیه آرمین

۸۱۰۱۹۷۶۴۸

۱۴۰۰ مهرماه

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 URL neo4j ایجاد شده را اجرا کردم.

The screenshot shows the Neo4j Browser interface. At the top, there's a terminal window with the command `$:play start`. Below it, the Neo4j logo is displayed. To the right, there are three cards: "Getting started with Neo4j Browser" (with "Get started" button), "Try Neo4j with live data" (with "Open guide" button), and "Cypher basics" (with "Start querying" button). At the bottom, another terminal window shows the command `$:server connect` and the response: "Connected to Neo4j" and "You are connected as user neo4j to neo4j://localhost:7687". A note says "Connection credentials are stored in your web browser."

در ادامه خروجی دستورات مختلف سایفر را مشاهده میکنید.

The screenshot shows the Neo4j Browser interface with the "Database Information" sidebar open. The "Use database" dropdown is set to "neo4j". In the main area, a terminal window shows the command `neo4j$ CREATE (:Album { Name : "Heavy as a Really Heavy Thing", Released : "1995" })`. The results pane shows a single node labeled "Album(1)" with a red circular badge containing "Heavy as a Really Heavy Thing". Below the results, a message says "Displaying 1 nodes, 0 relationships." Another terminal window at the bottom shows the command `neo4j$ CREATE (:Artist { Name : "Strapping Young Lad" })` and the message "Added 1 label, created 1 node, set 1 property, completed after 2 ms."

Database Information

Use database: neo4j

Node Labels: (175) Album, Artist, Movie, Person

Relationship Types: (256) ACTED IN, DIRECTED, FOLLOWS, PRODUCED, PLAYS IN, REVIEWED, WROTE

Property Keys: Name, Released, born, name, rating, released, roles, summary, tagline, title

Connected as: Username: neo4j, Roles: admin, PUBLIC, Admin: server user list, server user add, Disconnect: server disconnect

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 Time" })

Displaying 2 nodes, 0 relationships.

neo4j\$ CREATE (b:Album { Name : "Heavy as a Really Heavy Thing", Released : "1995" }) R...

Displaying 4 nodes, 8 relationships.

neo4j\$ MATCH (a:Artist),(b:Album),(p:Person) WHERE a.Name = "Strapping Young Lad" AND b.Na...

neo4j\$ CREATE (p:Person { Name: "Devin Townsend" })

Added 1 label, created 1 node, set 1 property, completed after 3 ms.

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_1ef5f5a7a	BTREE	NONUNIQUE	NODE	["Album"]	["Name"]	ONLINE
index_343af4e	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.

The screenshot shows the Neo4j Browser interface with the database set to 'neo4j'. In the top-right panel, the command `CREATE CONSTRAINT ON (m:Movie) ASSERT m.title IS UNIQUE` is run, resulting in the message 'Added 1 constraint, completed after 146 ms.'

In the bottom-right panel, a query `MATCH (a:Album {Name: "Somewhere in Time"}) USING INDEX a:Album(Name) RETURN a` is run, showing the result 'Album("Somewhere in Time")'.

The screenshot shows the Neo4j Browser interface with the database set to 'neo4j'. The top-right panel displays the schema with various indexes and constraints. One constraint is visible: `ON (movie:Movie) ASSERT (movie.title) IS UNIQUE`.

The screenshot shows the Neo4j Browser interface with the database set to 'neo4j'. The top-right panel shows the command `Create (:Movie {title: "Unforgiven"})`. An error message 'Neo.ClientError.Schema.ConstraintValidationFailed' is displayed, stating 'Node(97) already exists with label "Movie" and property "title" = "Unforgiven"'.

The bottom-right panel shows a graph visualization with nodes for 'The Bridges' (blue), 'Rescue' (orange), 'Jerry Maguire' (green), 'An Officer and a Gentleman' (red), and 'Born By Love' (purple).

The screenshot shows the Neo4j Browser interface. On the left, there's a sidebar titled "Database Information" with sections for "Use database" (set to "neo4j"), "Node Labels" (Album, Artist, Genre, Movie, Person), "Relationship Types" (ACTED IN, DIRECTED, FOLLOWS, PERFORMED ON, PLAYS IN, PRODUCED, RELEASED, REVIEWED, WROTE), and "Property Keys" (General, Name, Released, born, name, rating, released, roles, summary, tagline, title). The main area is titled "neo4j\$" and contains a graph visualization. A message at the top says "To enjoy the full Neo4j Browser experience, we advise you to use [Neo4j Browser Sync](#)". Below this, a query is shown: "MATCH (:Genre) RETURN n". The graph displays 115 nodes representing various genres as green circles of varying sizes. Nodes include Pop, Rock, Rap, Hip Hop, Hardcore, Drum & Bass, Comedy, Country, Classical, Folk, and others. A legend indicates node sizes based on their count: 1-10 (smallest), 11-100 (medium), and 101+ (largest). A search bar at the bottom right is set to "Displaying 115 nodes, 0 relationships." At the very bottom, a command line shows: "neo4j\$ LOAD CSV FROM 'https://www.quackit.com/neo4j/tutorial/genres.csv' AS line CREATE (:Genre {name: line[0]})".

The screenshot shows the Neo4j Browser interface with the following details:

- Database Information** sidebar:
 - Use database: neo4j
 - Node Labels: T(b2c), Album, Artist, Gene, Movie, Person, Track, album
 - Relationship Types: IS_A, ACTED_IN, DIRECTED, FOLLOWS, PERFORMED_ON, PLAYS_IN, PRODUCED, RELEASED, REVIEWED, WROTE
 - Property Keys: General, Length, Name, Released, TrackId, born, name, rating, released, roles, summary, tagline, title
- Neo4j\$** query panel:
 - Message: To enjoy the full Neo4j Browser experience, we advise you to use [Neo4j Browser Sync](#).
 - Query: `neo4j$ MATCH (n) DELETE n`
 - Error: **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\$** query panel:
 - Query: `neo4j$ MATCH (a:Artist {Name: "Iron Maiden"}), (b:Album {Name: "Powerslave"}) DELETE a, b`
 - Result: (no changes, no records)
- Bottom Panel**: Buttons for List, New, Warn, Close.

```
neo4j$ MATCH (n) DETACH DELETE n
Deleted 320 nodes, deleted 254 relationships, completed after 47 ms.

neo4j$ MATCH (a:Artist {Name: "Strapping Young Lad"}) DETACH DELETE a
Deleted 2 nodes, deleted 2 relationships, completed after 3 ms.
```

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
بررسیم.

```
neo4j$ MATCH (tom:Person) WHERE tom.name = "Tom Hanks" RETURN tom
```

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

The screenshot shows the Neo4j Browser interface running in a browser window. The URL is `localhost:7474/browser/`. The main area displays the results of the following Cypher query:

```
neo4j$ MATCH (cloudAtlas:Movie {title: "Cloud Atlas"}) RETURN cloudAtlas
```

The results are shown in a table format:

person.name
"Carrie-Anne Moss"
"Laurence Fishburne"
"Hugo Weaving"
"Lily Wachowski"
"Lana Wachowski"
"Joel Silver"

Below the table, a message indicates: "Started streaming 10 records after 1 ms and completed after 2 ms."

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.

The screenshot shows the Neo4j Browser interface running in a browser window. The URL is `localhost:7474/browser/`. The main area displays the results of the following Cypher query:

```
neo4j$ MATCH (people:Person) RETURN people.name LIMIT 10
```

The results are shown in a table format:

people.name
"Carrie-Anne Moss"
"Laurence Fishburne"
"Hugo Weaving"
"Lily Wachowski"
"Lana Wachowski"
"Joel Silver"

Below the table, a message indicates: "Started streaming 10 records after 1 ms and completed after 2 ms."

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.

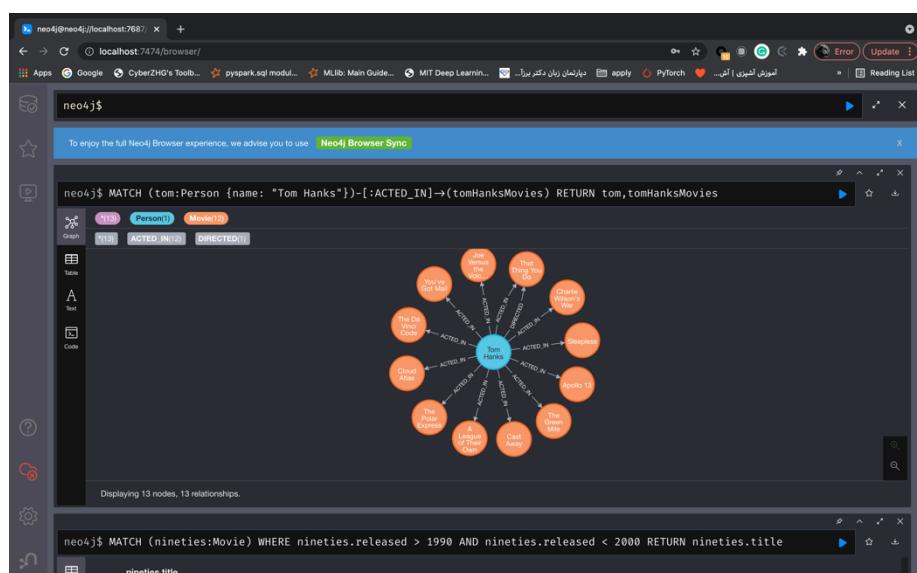
```
neo4j$ MATCH (nineties:Movie) WHERE nineties.released > 1990 AND nineties.released < 2000 RETURN nineties.title
```

nineties.title
"The Devil's Advocate"
"A Few Good Men"
"As Good as It Gets"
"What Dreams May Come"
"Snow Falling on Cedars"
"You've Got Mail"

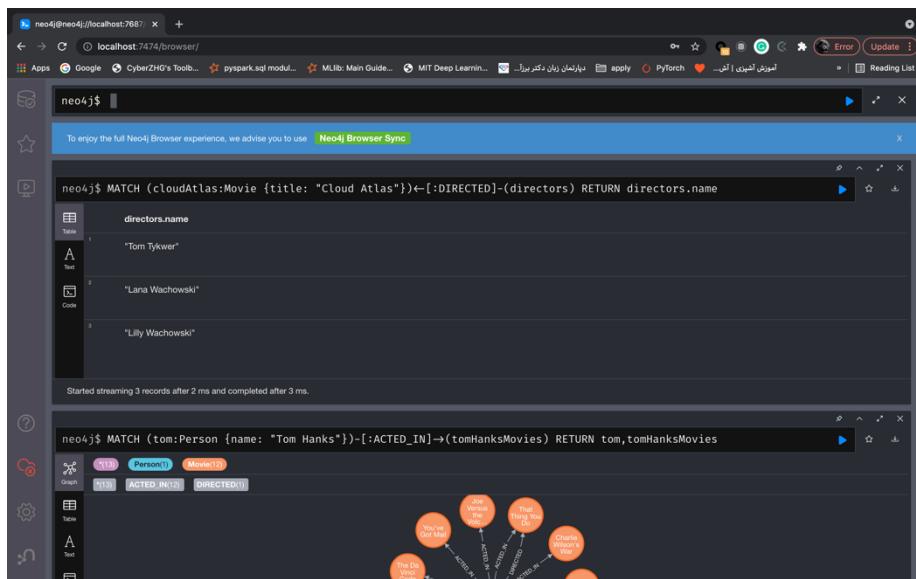
Started streaming 19 records after 1 ms and completed after 2 ms.

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

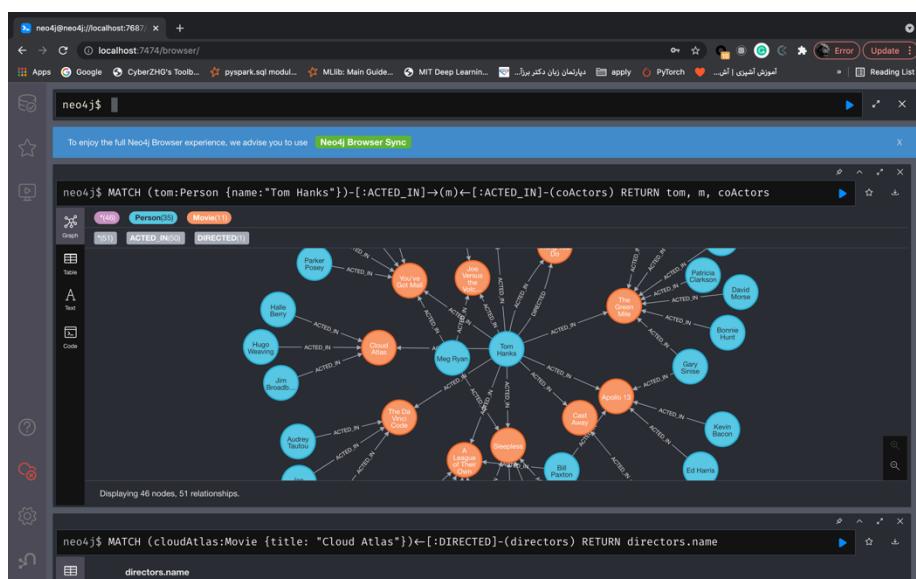
در دستور زیر میخواهیم تمام فیلم هایی که در آن ها است tom hanks باشد را بیابیم. برای این کار از ACTED_IN استفاده میکنیم و راس هایی را میباییم که با این رابطه با راس tom hanks relationship هستند.



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 رابطه DIRECTED به فیلم cloud Atlas متصل هستند را می یابیم.



در دستور زیر به دنبال افرادی هستیم که همراه با یک فیلم بازی کردنند. برای این کار ابتدا فیلم هایی که در آن ها بازی کرده و سپس افراد دیگری که با رابطه ACTED_IN به آن ها متصل شده اند را میباییم.



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.

The screenshot shows the Neo4j Browser interface with a query results table. The table has three columns: people.name, type[relatedTo], and relatedTo. The first row shows a result for "Tom Tykwer" with type "DIRECTED". The second row shows a result for "David Mitchell" with type "WROTE".

people.name	type[relatedTo]	relatedTo
"Tom Tykwer"	"DIRECTED"	{ "identity": 137, "start": 106, "end": 103, "type": "DIRECTED", "properties": { } } }
"David Mitchell"	"WROTE"	{ "identity": 140, "start": 107, "end": 105 }

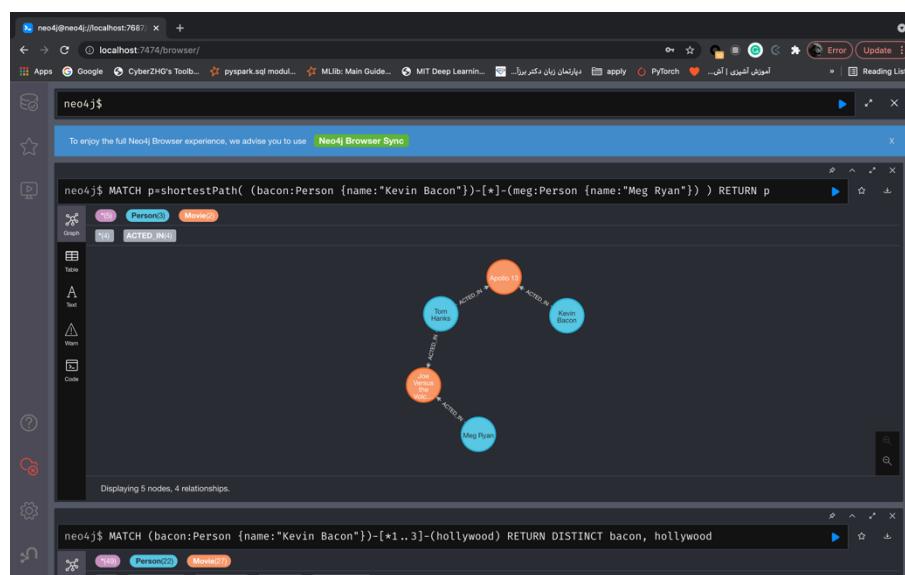
Started streaming 10 records after 1 ms and completed after 2 ms.

```
neo4j$ MATCH (people:Person)-[relatedTo]-(:Movie {title: "Cloud Atlas"}) RETURN people.name, type(relatedTo), relatedTo
```

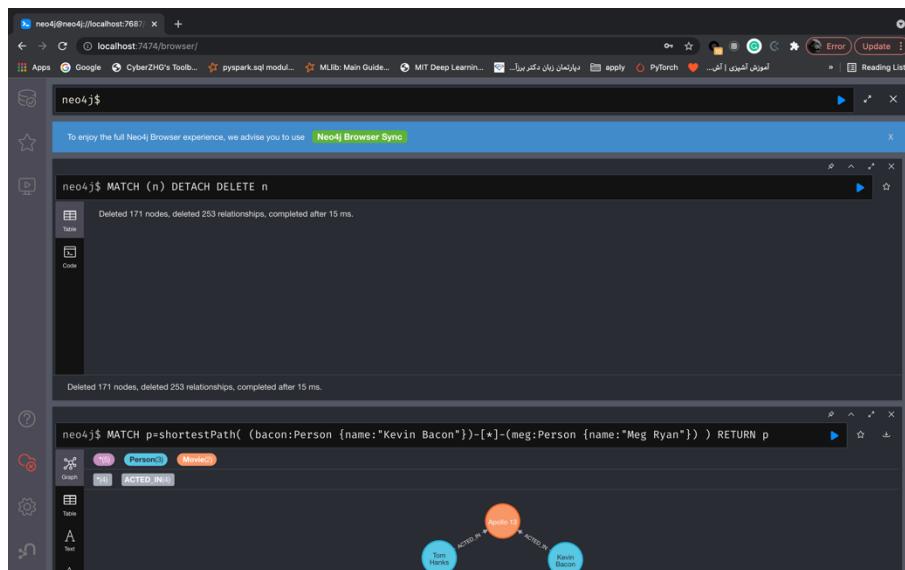
در دستورات زیر میخواهیم به چند سوال درباره گرافمان پاسخ دهیم.
در دستور زیر به دنبال تمام فیلم ها یا افرادی هستیم که حداقل ۳ hop از kevin bacon در گراف فاصله دارند. برای این کار از syntax ای به صورت $[*1..3]*$ استفاده میکنیم:

The screenshot shows the Neo4j Browser interface with a query results page. The main area displays a graph visualization of Kevin Bacon's connections. Nodes are represented as circles, and relationships as lines. Nodes include Kevin Bacon, Tim Robbins, Morgan Freeman, Edward Norton, Joaquin Phoenix, Christian Bale, and others. Relationships are labeled with actions like 'ACTED_IN', 'DIRECTED_BY', and 'WRITESHIP'. A sidebar on the left shows the query history and code editor. The top navigation bar includes tabs for Apps, Google, CyberZHGS's Tools..., pyspark.sql modul..., MLlib: Main Guide..., MIT Deep Learnin..., and PyTorch. The bottom status bar shows the command `neo4j\$ MATCH (people:Person)-[relatedTo](-:Movie {title: "Cloud Atlas"}) RETURN people.name, type(relatedTo), related...`.

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



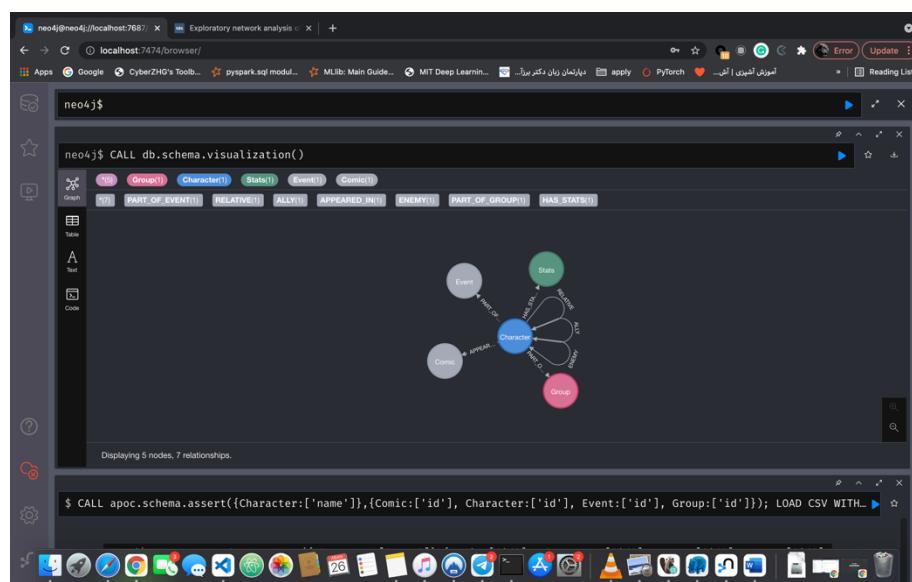
در دستورات بعدی به پاک کردن دیتاست میپردازیم. توجه داریم که راس ها تا وقتی که ارتباطات پاک نشوند، پاک نخواهد شد.



```

neo4j$ MATCH (n) RETURN count(*)
neo4j$ DETACH DELETE n
  
```

در ادامه به بررسی مقاله ذکر شده در صورت پروژه پرداختم. این مقاله سعی دارد گراف دیتابس دنیای marvel را آنالیز و بررسی کند. و در آخر با اعمال چند الگوریتم، دسته بندی خاصی را روی این گراف اعمال کند. ابتدا همانطور که در عکس زیر مشاهده میشود، داده ها را دانلود و شمای دیتا را نشان میدهیم.



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

The screenshot shows the Neo4j browser interface. At the top, the command `CALL apoc.meta.stats() YIELD labels return labels` is entered. Below the command, a table displays the following data:

	Labels
1	<pre>{ "Stats": 470, "Group": 92, "Event": 74, "Character": 1105, "Comic": 3875 }</pre>

Below the table, a message states: "Started streaming 1 records after 2 ms and completed after 1286 ms." In the bottom panel, a visualization of the graph structure is shown with nodes for Event and Stats, and relationships between them labeled RELATE.

در ادامه به آنالیز گراف میپردازیم.

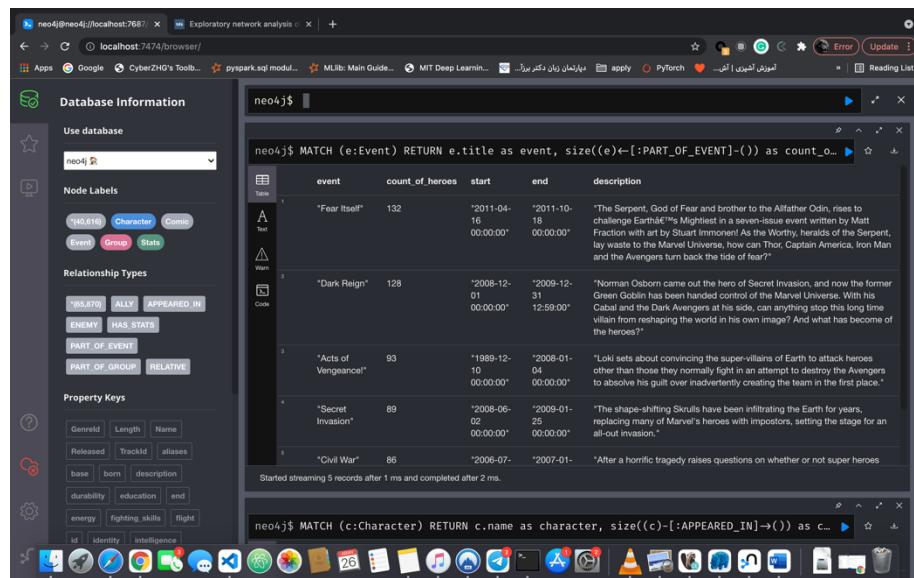
ابتدا به پیدا کردن کاراکتر هایی می پردازیم که بیشترین میزان تکرار را در comic ها دارند. برای اینکار جواب ها را به ۵ تا محدود میکنیم و ۵ کاراکتر پر تکرار را نمایش میدهیم.

The screenshot shows the Neo4j browser interface. At the top, the command `MATCH (c:Character) RETURN c.name as character, size((c)-[:APPEARED_IN]->()) as comics ORDER BY comics DESC LIMIT 5` is entered. Below the command, a table displays the following data:

character	comics
"Spider-Man (1602)"	3357
"Tony Stark"	2354
"Logan"	2098
"Steve Rogers"	2019
"Thor (Marvel: Avengers Alliance)"	1547

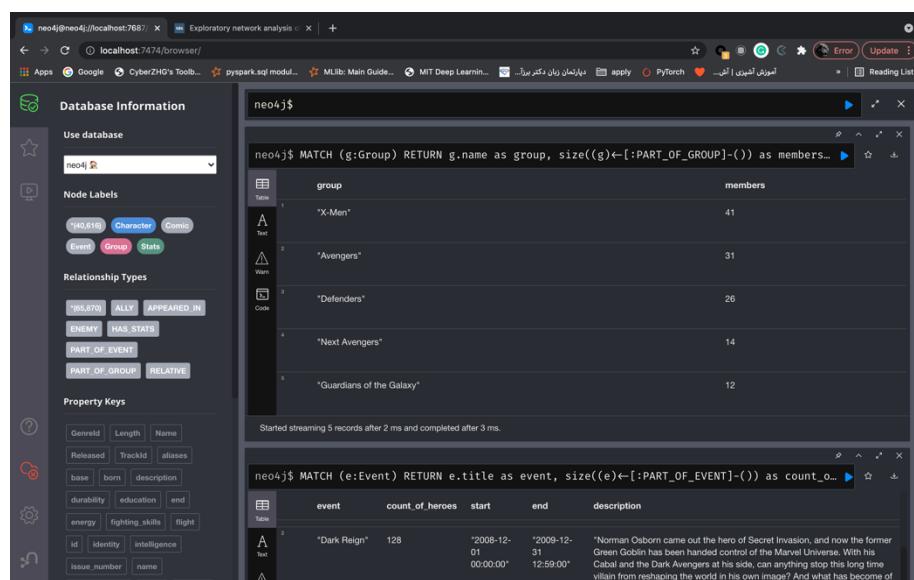
Below the table, a message states: "Started streaming 5 records after 2 ms and completed after 24 ms." In the bottom panel, the results of the `CALL apoc.meta.stats() YIELD labels return labels` query are shown, which is identical to the one in the first screenshot.

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.



```
neo4j$ MATCH (e:Event) RETURN e.title as event, size((e)-[:PART_OF_EVENT]--) as count_of_heroes, e.start, e.end, e.description
+-----+-----+-----+-----+
| event | count_of_heroes | start | end | description |
+-----+-----+-----+-----+
| "Fear Itself" | 132 | "2011-04-16 00:00:00" | "2011-10-18 00:00:00" | "The Serpent, God of Fear and brother to the Alttithor Odin, rises to challenge Earth's Mightiest in a seven-issue event written by Matt Fraction with art by Stuart Immonen. As the Worthy, heralds of the Serpent, lay waste to the Marvel Universe, how can Thor, Captain America, Iron Man and the Avengers turn back the tide of fear?" |
| "Dark Reign" | 128 | "2008-12-01 00:00:00" | "2009-12-31 12:59:00" | "Norman Osborn came out the hero of Secret Invasion, and now the former Green Goblin has been handed control of the Marvel Universe. With his Cabal and the Dark Avengers at his side, can anything stop this long time villain from reshaping the world in his own image? And what has become of the heroes?" |
| "Acts of Vengeance" | 93 | "1989-12-10 00:00:00" | "2008-01-04 00:00:00" | "Loki sets about convincing the super-villains of Earth to attack heroes other than those they normally fight in an attempt to destroy the Avengers to absolve his guilt over inadvertently creating the team in the first place." |
| "Secret Invasion" | 89 | "2008-06-02 00:00:00" | "2009-01-25 00:00:00" | "The shape-shifting Skrulls have been infiltrating the Earth for years, replacing many of Marvel's heroes with impostors, setting the stage for an all-out invasion." |
| "Civil War" | 86 | "2006-07-26 00:00:00" | "2007-01-01 00:00:00" | "After a horrific tragedy raises questions on whether or not super heroes" |
+-----+-----+-----+-----+
Started streaming 5 records after 1 ms and completed after 2 ms.
```

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.



```
neo4j$ MATCH (g:Group) RETURN g.name as group, size((g)-[:PART_OF_GROUP]--) as members
+-----+-----+
| group | members |
+-----+-----+
| "X-Men" | 41 |
| "Avengers" | 31 |
| "Defenders" | 26 |
| "Next Avengers" | 14 |
| "Guardians of the Galaxy" | 12 |
+-----+-----+
Started streaming 5 records after 2 ms and completed after 3 ms.
```



```
neo4j$ MATCH (e:Event) RETURN e.title as event, size((e)-[:PART_OF_EVENT]--) as count_of_heroes, e.start, e.end, e.description
+-----+-----+-----+-----+
| event | count_of_heroes | start | end | description |
+-----+-----+-----+-----+
| "Dark Reign" | 128 | "2008-12-01 00:00:00" | "2009-12-31 12:59:00" | "Norman Osborn came out the hero of Secret Invasion, and now the former Green Goblin has been handed control of the Marvel Universe. With his Cabal and the Dark Avengers at his side, can anything stop this long time villain from reshaping the world in his own image? And what has become of the heroes?" |
+-----+-----+-----+-----+
```

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

```

neo4j$ MATCH (c1:Character)-[:PART_OF_GROUP]-(g:Group)-[:PART_OF_GROUP]-(c2:Character) ...
neo4j$ MATCH (g:Group) RETURN g.name as group, size((g)-[:PART_OF_GROUP]--) as members...
  
```

character1	character2	group
"Logan"	"Sabretooth (House of M)"	"X-Men"
"Logan"	"Mystique (House of M)"	"X-Men"
"CAIN MARKO JUGGERNAUT"	"Logan"	"X-Men"
"CAIN MARKO JUGGERNAUT"	"Storm (Marvel Heroes)"	"X-Men"
"Rogue (X-Men: Battle of the Atom)"	"Warren Worthington III"	"X-Men"

group	members
"X-Men"	41
"Avengers"	31

حال میخواهیم بینیم چه کاراکتر هایی از Yugoslavia ظهرور کرده اند. برای اینکار از عملگر contains استفاده کرده و کاراکتر هایی که place of origin Yugoslavia باشد را جدا میکنیم.

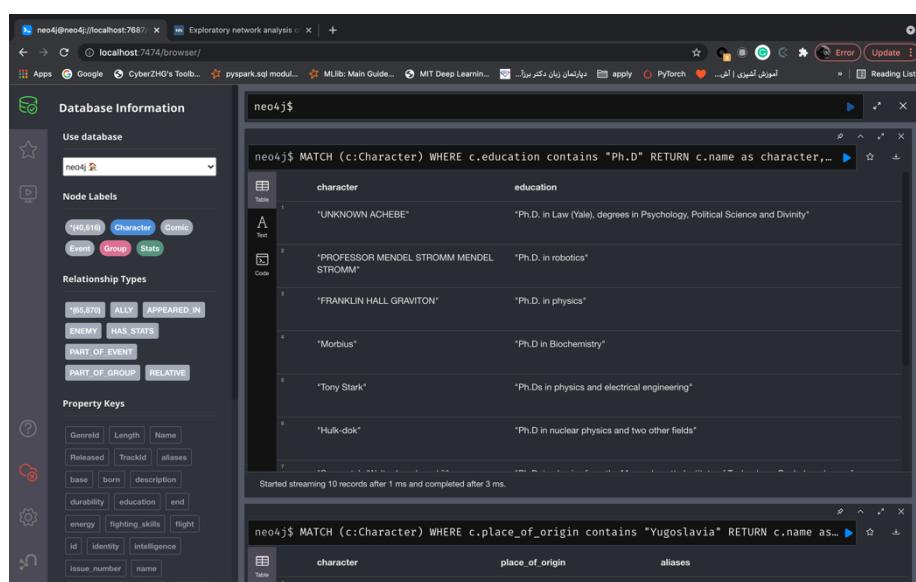
```

neo4j$ MATCH (c:Character) WHERE c.place_of_origin contains "Yugoslavia" RETURN c.name as...
neo4j$ MATCH (c1:Character)-[:PART_OF_GROUP]-(g:Group)-[:PART_OF_GROUP]-(c2:Character) ...
  
```

character	place_of_origin	aliases
"Purple Man"	"Rijeka, Yugoslavia"	"Killgrave the Purple Man, Killy"
"Abomination (Ultimate)"	"Zagreb, Yugoslavia"	"Agent R-7, the Ravager of Worlds"

character1	character2	group
"Logan"	"Sabretooth (House of M)"	"X-Men"
"Logan"	"Mystique (House of M)"	"X-Men"
"CAIN MARKO JUGGERNAUT"	"Logan"	"X-Men"

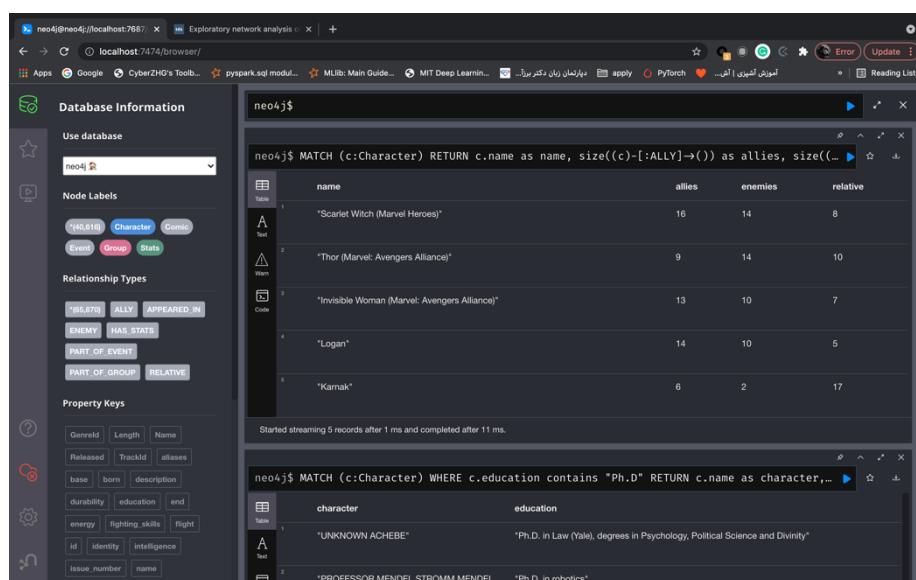
برای آنکه کمی بیشتر با کاراکتر ها آشنا شویم، به دنبال کاراکتر هایی میگردیم که مدرک Ph.D شان را گرفته اند و ۱۰ نفر آن ها را نمایش میدهیم.



```
neo4j$ MATCH (c:Character) WHERE c.education contains "Ph.D" RETURN c.name as character, c.education
+-----+-----+
| character | education |
+-----+-----+
| "UNKNOWN ACHEBE" | "Ph.D. in Law (Yale), degrees in Psychology, Political Science and Divinity" |
| "PROFESSOR MENDEL STROMM MENDEL STROMM" | "Ph.D. in robotics" |
| "FRANKLIN HALL GRAVITON" | "Ph.D. in physics" |
| "Morbius" | "Ph.D. in Biochemistry" |
| "Tony Stark" | "Ph.D.s in physics and electrical engineering" |
| "Hulk-dok" | "Ph.D. in nuclear physics and two other fields" |
+-----+-----+
Started streaming 10 records after 1 ms and completed after 3 ms.

neo4j$ MATCH (c:Character) WHERE c.place_of_origin contains "Yugoslavia" RETURN c.name as character, c.place_of_origin, c_aliases
+-----+-----+-----+
| character | place_of_origin | aliases |
+-----+-----+-----+
| "Scarlet Witch (Marvel Heroes)" | "Yugoslavia" | 16 |
| "Thor (Marvel: Avengers Alliance)" | "Yugoslavia" | 9 |
| "Invisible Woman (Marvel: Avengers Alliance)" | "Yugoslavia" | 13 |
| "Logan" | "Yugoslavia" | 14 |
| "Karnak" | "Yugoslavia" | 6 |
+-----+-----+-----+
```

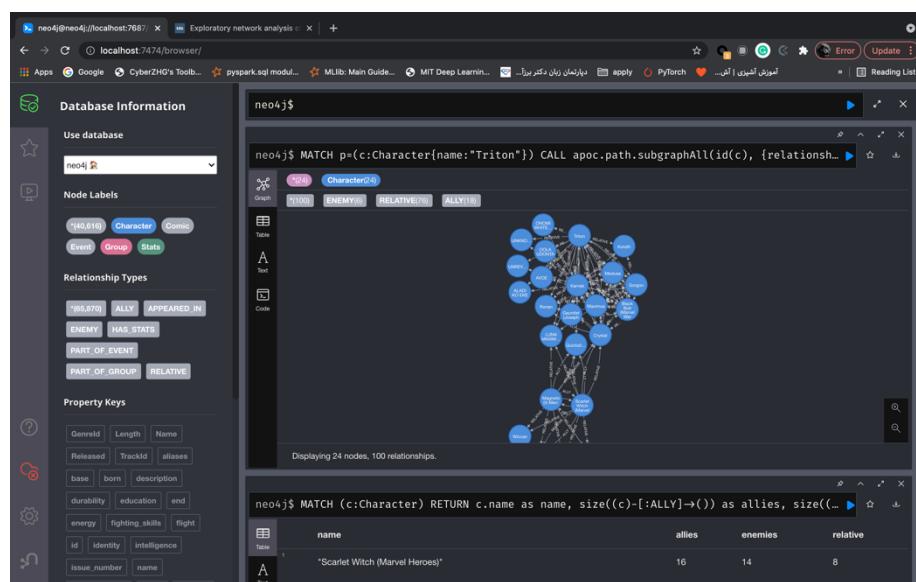
در ادامه به بررسی و آنالیز جامعه ها و community های بستگان و متعددان میپردازیم. برای اینکار ابتدا برای هر کاراکتر، تعداد بستگان، متعددان و دشمنان آن را پیدا میکنیم. این کار با بررسی روابط ENEMY, ALLY, RELATIVE انجام می شود.



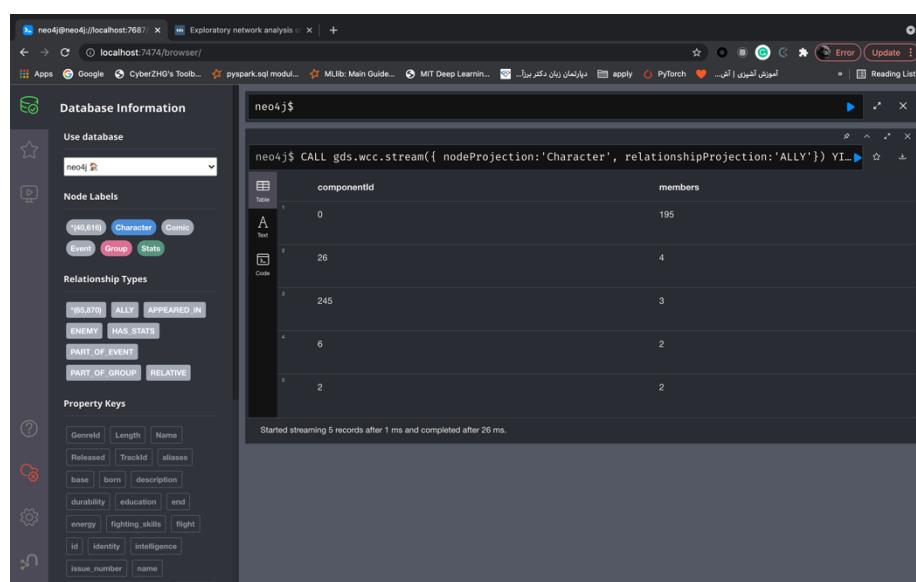
```
neo4j$ MATCH (c:Character) RETURN c.name as name, size((c)-[:ALLY]-()) as allies, size((c)-[:ENEMY]-()) as enemies, size((c)-[:RELATIONSHIP]-()) as relatives
+-----+-----+-----+-----+
| name | allies | enemies | relatives |
+-----+-----+-----+-----+
| "Scarlet Witch (Marvel Heroes)" | 16 | 14 | 8 |
| "Thor (Marvel: Avengers Alliance)" | 9 | 14 | 10 |
| "Invisible Woman (Marvel: Avengers Alliance)" | 13 | 10 | 7 |
| "Logan" | 14 | 10 | 5 |
| "Karnak" | 6 | 2 | 17 |
+-----+-----+-----+-----+
Started streaming 5 records after 1 ms and completed after 11 ms.

neo4j$ MATCH (c:Character) WHERE c.education contains "Ph.D" RETURN c.name as character, c.education
+-----+-----+
| character | education |
+-----+-----+
| "UNKNOWN ACHEBE" | "Ph.D. in Law (Yale), degrees in Psychology, Political Science and Divinity" |
| "PROFESSOR MENDEL STROMM MENDEL" | "Ph.D. in robotics" |
+-----+-----+
```

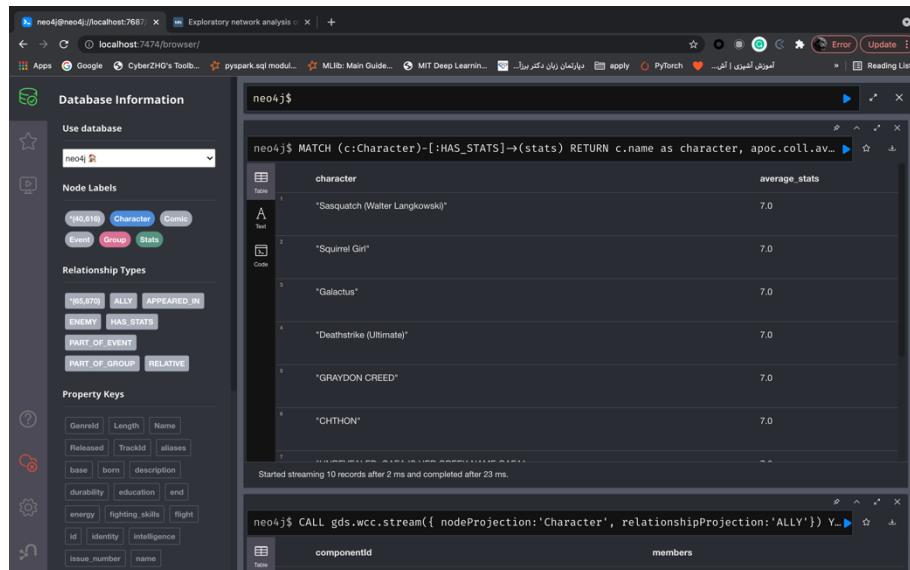
برای آنکه جامعه بستگان یک کاراکتر را بررسی کنیم میتوانیم از `apoc.path.subgraphAll()` استفاده کنیم و گراف آن را ببینیم.



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



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

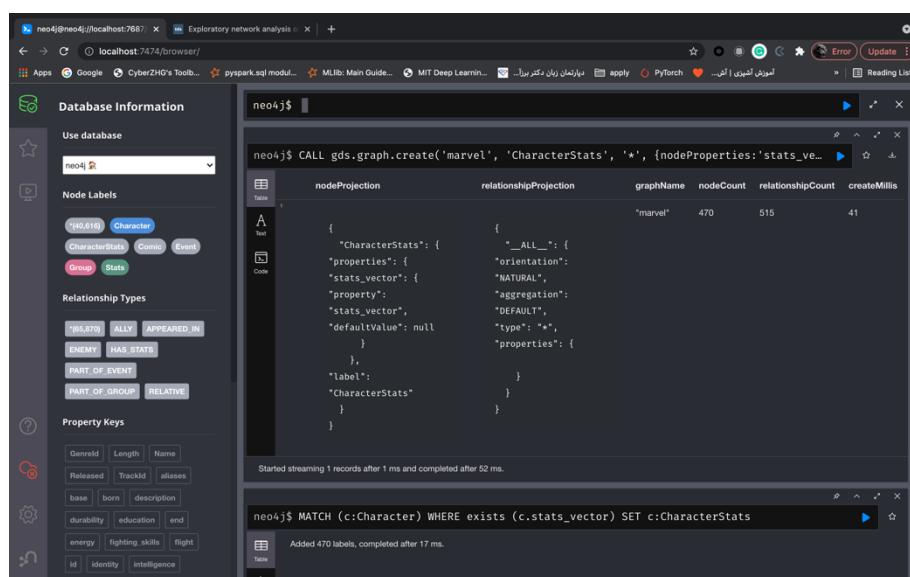


```
neo4j$ MATCH (c:Character)-[:HAS_STATS]-(stats) RETURN c.name as character, apoc.coll.average(stats) as average_stats
neo4j$ CALL gds.wcc.stream({ nodeProjection:'Character', relationshipProjection:'ALLY' }) Y-
```

character	average_stats
"Sasquatch (Walter Langkowski)"	7.0
"Squirrel Girl"	7.0
"Galactus"	7.0
"Deathstrike (Ultimate)"	7.0
"GRAYDON CREED"	7.0
"CHTHON"	7.0

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 مقایسه میشود. علاوه بر ویرگی هایی که در دیتاست هست، یک ویرگی 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.



```
neo4j$ CALL gds.graph.create('marvel', 'CharacterStats', '*', {nodeProperties:'stats_vector'}) Y-
neo4j$ MATCH (c:Character) WHERE exists (c.stats_vector) SET c:CharacterStats
neo4j$
```

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	createMillis
{ "CharacterStats": { "properties": { "stats_vector": { "property": "stats_vector", "label": "CharacterStats" } } } }	{ "_ALL_": { "orientation": "NATURAL", "aggregation": "DEFULT", "type": "+", "properties": {} } }	'marvel'	470	515	41

حال الگوریتم knn را با topK ۱۵ و sampleRate ۰.۸ 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.

```
neo4j$ CALL gds.beta.knn.mutate('marvel', {nodeWeightProperty:'stats_vector', sampleRate:0.8, topK:15, mutateProperty:'score', mutateRelationshipType:'SIMILAR'})
```

	createMillis	computeMillis	mutateMillis	postProcessingMillis	nodesCompared	relationshipsWritten	similarityDistribution
Total	0	238	41	-1	470	7050	{ "p1": 0.25000095367431 "max": 1.00000667572021 "p5": 0.3333301544189 "p9": 0.5000286102294 "p50": 0.5000286102294 "p95": 1.00000667572021 "p10": 0.3333301544189 }

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 هنگام بررسی ساختار باید وزن رابطه ها را در نظر بگیرد.

```
neo4j$ CALL gds.louvain.write('marvel', {relationshipTypes:['SIMILAR'], relationshipWeightProperty:'weight'})
```

	writeMillis	nodePropertiesWritten	modularity	modularities	ranLevels	communityCount	communityDistribution
Total	31	470	0.6177651219242726	[0.5417975225974798, 0.6177651219242726]	2	8	{ "p99": 78 "min": 28 "max": 78 "mean": 58.75 "p99": 77 "p50": 61 "p999": 7 "p95": 78 "p75": 77 }


```
neo4j$ CALL gds.beta.knn.mutate('marvel', {nodeWeightProperty:'stats_vector', sampleRate:0.8})
```

	createMillis	computeMillis	mutateMillis	postProcessingMillis	nodesCompared	relationshipsWritten	similarityDistribution
Total	0	238	41	-1	470	7050	{ "p1": 0.25000095367431 "max": 1.00000667572021 "p5": 0.3333301544189 "p9": 0.5000286102294 "p50": 0.5000286102294 "p95": 1.00000667572021 "p10": 0.3333301544189 }

حال در هر متوسط هر skill را محاسبه میکنیم.

```

neo4j$ MATCH (c:Character)-[r:HAS_STATS]-(stats) RETURN c.louvain as community, count(*) as members, r.fighting_skills as fighting_skills, r.durability as durability, r.energy as energy, r.intelligence as intelligence, r.speed as speed
    
```

community	members	fighting_skills	durability	energy	intelligence	speed
1	59	29	4.517241379310344	5.0344827586206805	4.689655172413794	4.482758620699565
2	362	77	2.6883116883116878	2.2467532467532467	1.7012987012987015	2.818181818181818
3	298	77	3.7012987012987018	3.0389610389610398	2.4025974025974026	2.974025974025974
4	131	75	4.5333333333333334	4.64	3.5599999999999996	3.8133333333333326
5	235	78	3.6025641025641026	4.076923076923075	2.5512620512820515	3.2692307692307705
6	67	61	4.55737704918033	6.01639342622951	5.2950819672131155	4.311475409836066

Started streaming 8 records after 1 ms and completed after 11 ms.

```

neo4j$ CALL gds.louvain.write('marvel', {relationshipTypes:['SIMILAR'], relationshipWeightProperty: 'APPEARED_IN', writeMillis: 1000, nodePropertiesWritten: true})
    
```

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

```

neo4j$ MATCH (c:Character)-[r:HAS_STATS]-(stats) RETURN c.labelPropagation as community, count(*) as members, r.fighting_skills as fighting_skills, r.durability as durability, r.energy as energy, r.intelligence as intelligence, r.speed as speed
    
```

community	members	fighting_skills	durability	energy	intelligence	speed
1	444	9	4.333333333333333	5.555555555555545	4.6666666666666667	4.444444444444445
2	216	76	3.0526315789473677	2.631578947368421	1.97368421052631593	2.184210526315
3	19	47	4.234042553191491	3.1914893617021276	2.2765957446808507	3.0212765957446814
4	270	24	4.708333333333333	5.2083333333333335	3.833333333333344	4.375
5	114	98	3.326530612244899	3.530612244897959	2.44897951836735	3.1632653061224496
6	215	17	4.470588235294118	3.4705882352941173	2.6470588235294112	3.294117647058823

Started streaming 16 records after 1 ms and completed after 6 ms.

```

neo4j$ CALL gds.labelPropagation.write('marvel', {relationshipTypes:['SIMILAR'], relationshipWeightProperty: 'APPEARED_IN', writeMillis: 1000, nodePropertiesWritten: true})
    
```

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