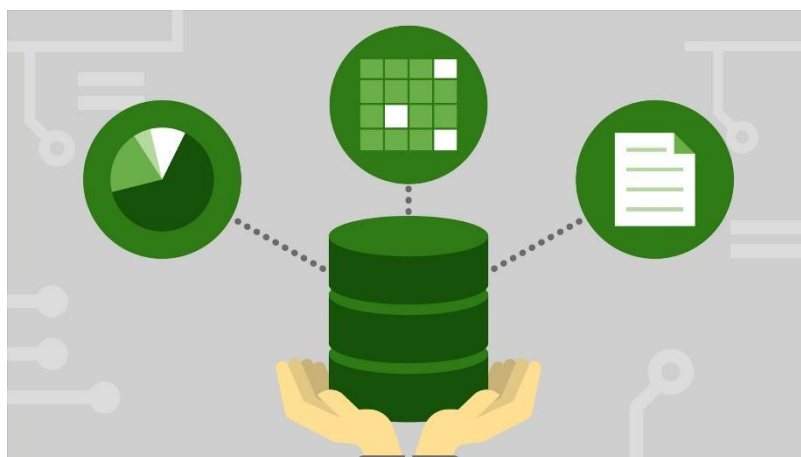


به نام خدا



دانشگاه تهران  
پردیس دانشکده‌های فنی  
دانشکده برق و کامپیوتر



### آزمایشگاه پایگاه داده

دستور کار شماره 3

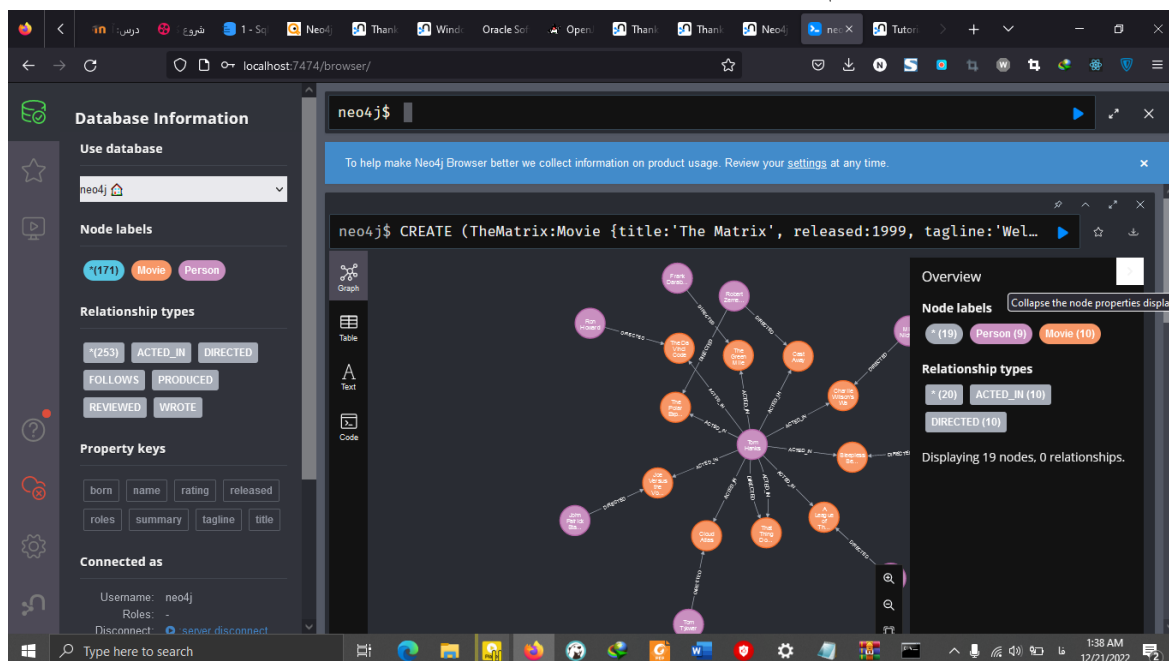
نگار مرادی

810198543

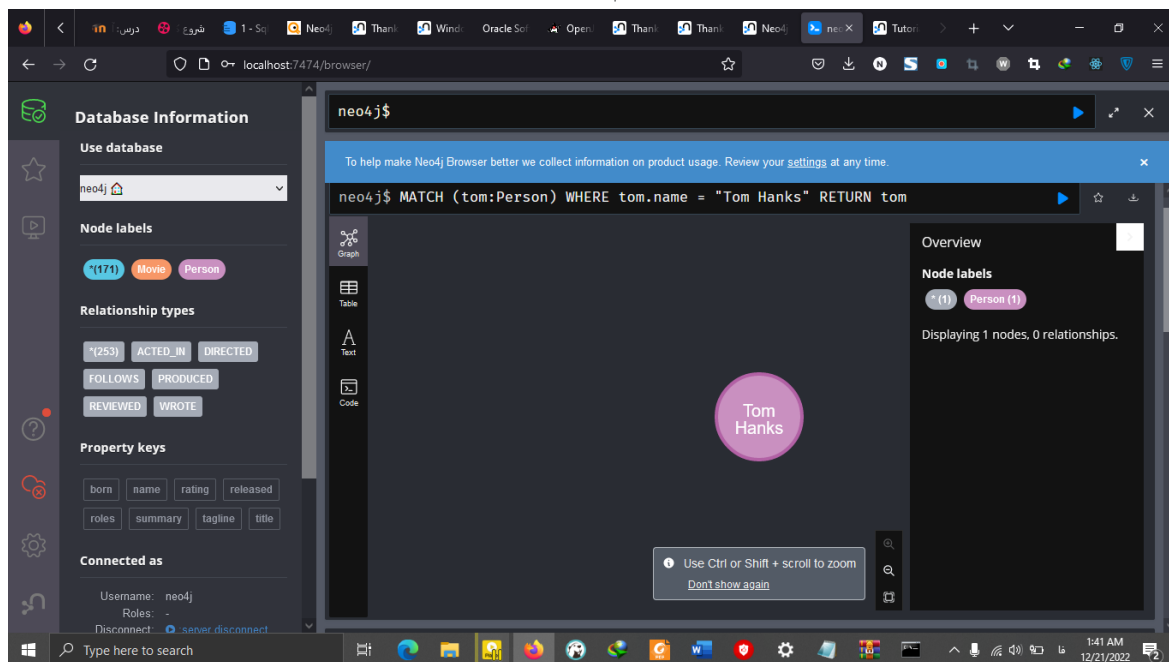
مهر 1401

## گزارش دستورکار انجام شده

طبق دستورات دیتابیس را لود کردیم



دستور بعدی ادمی با نام tom hanks را پیدا میکند. در این دستور تمام person هایی که نام آن ها tom hanks هست را پیدا میکند. و در انتها یک نود person که name آن تام هنکس است را باز میگرداند.



همچین جدول آن را می توانیم مشاهده کنیم:

The screenshot shows the Neo4j Browser interface. On the left, the 'Database Information' sidebar is visible, showing the database 'neo4j' and various node labels like 'Movie' and 'Person'. The main area displays a Cypher query: `neo4j$ MATCH (tom:Person) WHERE tom.name = "Tom Hanks" RETURN tom`. The result is shown in a table format, displaying a single record for Tom Hanks with properties like 'identity', 'labels', 'properties', 'born', 'name', and 'elementId'.

پیدا کردن فیلم با نام Cloud Atlas مانند مثال قبل تمام نودهای movie که title آن ها برابر cloud atlas است را پیدا میکنیم و آن نود را بازمیگردانیم.

The screenshot shows the Neo4j Browser interface. On the left, the 'Database Information' sidebar is visible, showing the database 'neo4j' and various node labels like 'Movie' and 'Person'. The main area displays a Cypher query: `neo4j$ MATCH (cloudAtlas:Movie {title: "Cloud Atlas"}) RETURN cloudAtlas`. The result is shown in a table format, displaying a single record for Cloud Atlas with properties like 'identity', 'labels', 'properties', 'tagline', 'title', 'released', and 'elementId'.

پیدا کردن اسم 10 نفر تا اندازه 10 نفر person return میکنیم. 10 خروجی اول تابع match را return میکنیم.

The screenshot shows the Neo4j Browser interface. On the left, the 'Database Information' sidebar is visible, showing the database 'neo4j' and various node labels like 'Movie' and 'Person'. The main area displays a Cypher query:

```
1 MATCH (people:Person)
2 RETURN people.name LIMIT 10
```

The results are shown in a table with the header 'people.name' and 10 rows of names:

people.name
"Keanu Reeves"
"Carrie-Anne Moss"
"Laurence Fishburne"
"Hugo Weaving"
"Lilly Wachowski"
"Lana Wachowski"

At the bottom, a status message reads: 'Started streaming 10 records after 2 ms and completed after 3 ms.'

پیدا کردن فیلم هایی که در release 1990 شده اند.

The screenshot shows the Neo4j Browser interface. The main area displays a Cypher query:

```
1 MATCH (nineties:Movie)
2 WHERE nineties.release > 1990 AND nineties.release < 2000
3 RETURN nineties.title
```

The results are shown in a table with the header 'nineties.title' and 6 rows of movie titles:

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

این query تمام نودهای movie که سال تولیدشان (release) بین 1990 و 2000 است را پیدا کرده و تایتل آن ها را return میکند.

[illegible]

The screenshot displays the Neo4j browser interface. On the left sidebar, the 'Database Information' section shows the database name 'neo4j'. Below it, 'Node labels' are listed as '(171) Movie' and 'Person'. The 'Relationship types' section shows 'ACTED\_IN' and 'DIRECTED'. The 'Property keys' section lists 'born', 'name', 'rating', 'released', 'roles', 'summary', 'tagline', and 'title'. The 'Connected as' section shows the user 'neo4j' is connected to the 'server'.

The main area shows a Cypher query:
 

```
1 MATCH (tom:Person {name: 'Tom Hanks'})-[:ACTED_IN]->(tomHanksMovies:Movie)
2 RETURN tom,tomHanksMovies
```

 The query results are displayed in a table with two columns: 'tom' and 'tomHanksMovies'. The 'tom' column contains a node with properties: 'identity': 71, 'labels': ['Person'], 'properties': {'born': 1956, 'name': 'Tom Hanks'}, and 'elementId': '71'. The 'tomHanksMovies' column contains a node with properties: 'identity': 144, 'labels': ['Movie'], 'properties': {'tagline': 'Houston, we have a problem.', 'title': 'Apollo 13', 'released': 1995}, and 'elementId': '144'.

At the bottom, the 'Create' section shows the command:
 

```
$ :play movie graph
```

 The output shows the command being executed and the results of the query.

## لیست کارگردان های فیلم Cloud Atlas

The screenshot shows the Neo4j Desktop interface. On the left, the 'Database Information' sidebar is visible, showing the database name 'neo4j' and various node labels like 'Movie' and 'Person'. The main workspace displays a Cypher query:

```
1 MATCH (cloudAtlas:Movie {title: "Cloud Atlas"})<--[:DIRECTED]-(directors)
2 RETURN directors.name
```

The query results show a list of directors: "Tom Tykwer", "Lana Wachowski", and "Lily Wachowski". Below the query, there is a section titled 'The Movie Graph' with a 'Create' button and a code block containing Cypher commands to create a movie graph, including nodes for 'The Matrix', 'Keanu Reeves', 'Carrie-Anne Moss', 'Laurence Fishburne', 'Hugo Weaving', 'Lilly Wachowski', 'Lana Wachowski', and 'Joel Silver'.

این دستور اسم تمام نودهایی که رابطه directed با نود movie که نامش cloud atlas دارد را بازمیگرداند. تمام فیلم هایی که نام هنکس در آن ها بازی کرده و تمام بازیگرانی که در این فیلم ها بازی کرده اند.

The screenshot shows the Neo4j Desktop interface. The main workspace displays a Cypher query:

```
1 MATCH (tom:Person {name: "Tom Hanks"})<--[:ACTED_IN]-(m)<--[:ACTED_IN]-(coActors)
2 RETURN tom, m, coActors
```

The query results show a complex graph visualization with many nodes and edges, representing the relationships between Tom Hanks, movies, and other actors. Below the query, there is a section titled 'The Movie Graph' with a 'Create' button and a code block containing Cypher commands to create a movie graph, including nodes for 'The Matrix', 'Keanu Reeves', 'Carrie-Anne Moss', and 'Laurence Fishburne'.

The screenshot shows the Neo4j Browser interface. On the left, the 'Database Information' sidebar is visible, showing the database 'neo4j' and various filters. The main area displays a Cypher query:

```
1 MATCH (tom:Person {name:"Tom Hanks"})-[:ACTED_IN]->(m)-[:ACTED_IN]-(coActors)
2 RETURN tom, m, coActors
```

The results are shown in a table with three columns: 'tom', 'm', and 'coActors'. Each column contains a JSON object representing a node in the graph. The 'tom' node is a Person with identity 71, born in 1956, and name 'Tom Hanks'. The 'm' node is a Movie with identity 144, title 'Apollo 13', and released in 1995. The 'coActors' node is a Person with identity 145, born in 1950, and name 'Ed Harris'. Below the table, a status message indicates 'Started streaming 39 records after 2 ms and completed after 16 ms'.

در این query نودهایی که با نام هنکس رابطه acted\_in دارند و همچنین هر نود دیگه ای که با این فیلم ها رابطه acted\_in دارد را میگیریم.

نام تمام افرادی که به فیلم cloud atlas ارتباط دارند و نوع ارتباطشان.

The screenshot shows the Neo4j Browser interface. On the left, the 'Database Information' sidebar is visible, showing the database 'neo4j' and various filters. The main area displays a Cypher query:

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

The results are shown in a table with three columns: 'people.name', 'type(relatedTo)', and 'relatedTo'. The 'people.name' column contains the name 'Tom Hanks'. The 'type(relatedTo)' column contains the relationship type 'ACTED\_IN'. The 'relatedTo' column contains a JSON object representing a Movie node with identity 137, start in 71, end in 185, type 'ACTED\_IN', and roles ['Zachry', 'Dr. Henry Goose', 'Isaac Sachs', 'Dermot Hoggins']. Below the table, a status message indicates 'Started streaming 10 records after 2 ms and completed after 11 ms'.

تمام نام نودهایی که رابطه relatedTo با فیلمی که تایتل cloud atlas دارد دارند و نوع رابطه شان را با استفاده از type باز میگردانیم.

Query که در آن هر نودی که در گراف فاصله ای بین 1 تا 3 دارد از نود Kevin Bacon را برمیگردانیم. (چه فیلم باشد چه person) فاصله 1 تا 3 را با

[\*1..3]

نشان میدهیم.

The screenshot shows the Neo4j Browser interface. On the left, the 'Database Information' sidebar is visible, showing the database name 'neo4j', node labels 'Movie' and 'Person', and relationship types 'ACTED\_IN', 'DIRECTED', 'FOLLOWS', 'PRODUCED', 'REVIEWED', and 'WROTE'. The main area displays a Cypher query:

```
1 MATCH (bacon:Person {name:'Kevin Bacon'})-[*1..3]-(hollywood)
2 RETURN DISTINCT bacon, hollywood
```

The query is executed, and the result is visualized as a graph. The graph shows a central node 'Kevin Bacon' (Person) connected to several other nodes, including 'The Matrix' (Movie) and 'Keanu Reeves' (Person). The graph is titled 'The Movie Graph'.

The screenshot shows the Neo4j Browser interface with the same Cypher query as above. The query is executed, and the result is displayed as a JSON array of objects. The first object represents the relationship between 'Kevin Bacon' and 'The Matrix'.

```
{
  "identity": 19,
  "labels": [
    "Person"
  ],
  "properties": {
    "born": 1958,
    "name": "Kevin Bacon"
  },
  "elementId": "19"
}, {
  "identity": 144,
  "labels": [
    "Movie"
  ],
  "properties": {
    "tagline": "Houston, we have a problem.",
    "title": "Apollo 13",
    "released": 1995
  },
  "elementId": "144"
}
```

The second object represents the relationship between 'Kevin Bacon' and 'Keanu Reeves'.

```
{
  "identity": 19,
  "labels": [
    "Person"
  ],
  "properties": {
    "born": 1958,
    "name": "Kevin Bacon"
  },
  "elementId": "19"
}, {
  "identity": 137,
  "labels": [
    "Person"
  ],
  "properties": {
    "born": 1964,
    "name": "Keanu Reeves"
  },
  "elementId": "137"
}
```

The interface also shows the 'play movie graph' button and the 'Create' button.



کوتاه ترین مسیر بین دو نود Kevin Bacon , Meg Ryan person را باز میگردانیم که ممکن است یک نود و یا یک رابطه باشد.

The screenshot shows the Neo4j Desktop interface. On the left, the 'Database Information' sidebar is visible. The main editor displays a Cypher query:

```
1 MATCH p=shortestPath(
2   (bacon:Person {name:"Kevin Bacon"})-[*]-(meg:Person {name:"Meg Ryan"})
3 )
4 RETURN p
```

The query is executed, and the result is a graph visualization showing the shortest path between Kevin Bacon and Meg Ryan. The path consists of the following nodes and relationships:

- Kevin Bacon (Person) -> ACTED\_IN -> Top Gun (Movie) -> ACTED\_IN -> Meg Ryan (Person)

The interface also shows a 'play movie graph' button and a 'The Movie Graph' section at the bottom.

ابتدا دو نود person مورد نظر را پیدا می‌کنیم و سپس مسیرهای بین این دو نود را پیدا می‌کنیم (تمام مسیرها \* ) و در انتها shortest path این مسیرها را در نظر می‌گیریم.  
بعد از اتمام کارمان این دیتاست را پاک می‌کنیم با استفاده از دستور زیر

The screenshot shows the Neo4j Desktop interface. The main editor displays a Cypher query:

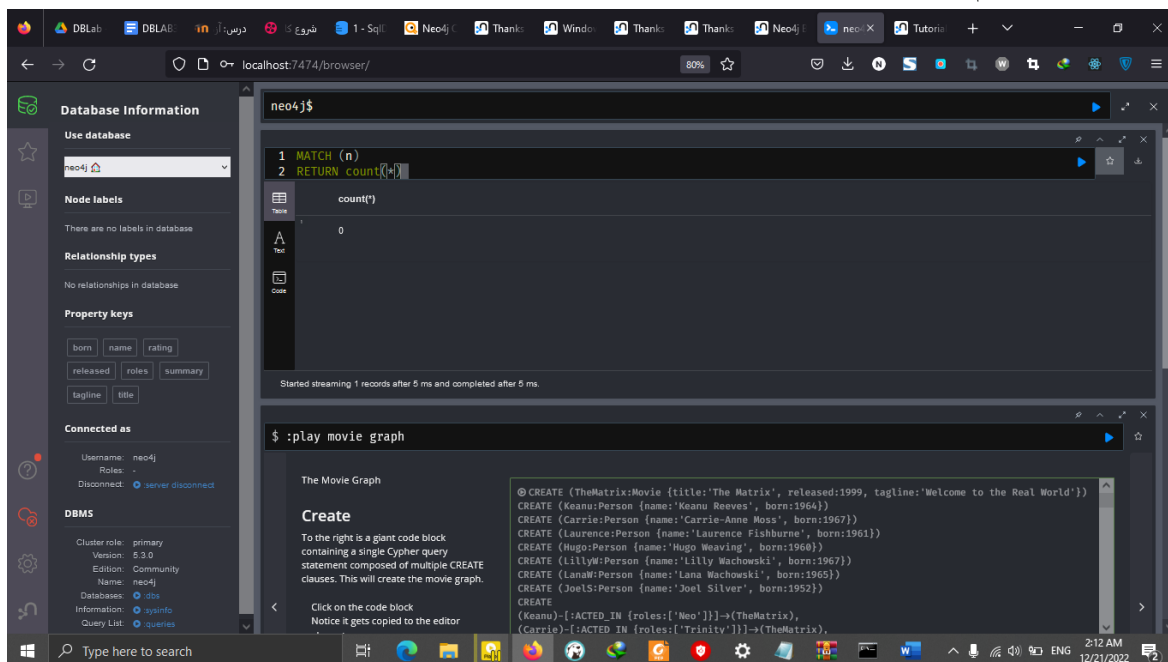
```
1 MATCH (n)
2 DETACH DELETE n
```

The query is executed, and the result is a message: 'Deleted 171 nodes, deleted 253 relationships, completed after 243 ms.' Below the query, there is a 'Create' section with a large block of Cypher code for creating a movie graph:

```
@ CREATE (TheMatrix:Movie {title:'The Matrix', released:1999, tagline:'Welcome to the Real World'})
CREATE (Keanu:Person {name:'Keanu Reeves', born:1964})
CREATE (Carrie:Person {name:'Carrie-Anne Moss', born:1967})
CREATE (Laurence:Person {name:'Laurence Fishburne', born:1961})
CREATE (Hugo:Person {name:'Hugo Weaving', born:1960})
CREATE (LillyW:Person {name:'Lilly Wachowski', born:1967})
CREATE (LanaW:Person {name:'Lana Wachowski', born:1965})
CREATE (JoelS:Person {name:'Joel Silver', born:1952})
CREATE
(Keanu)-[:ACTED_IN {roles:['Neo']}]->(TheMatrix),
(Carrie)-[:ACTED_IN {roles:['Trinity']}]>(TheMatrix),
```

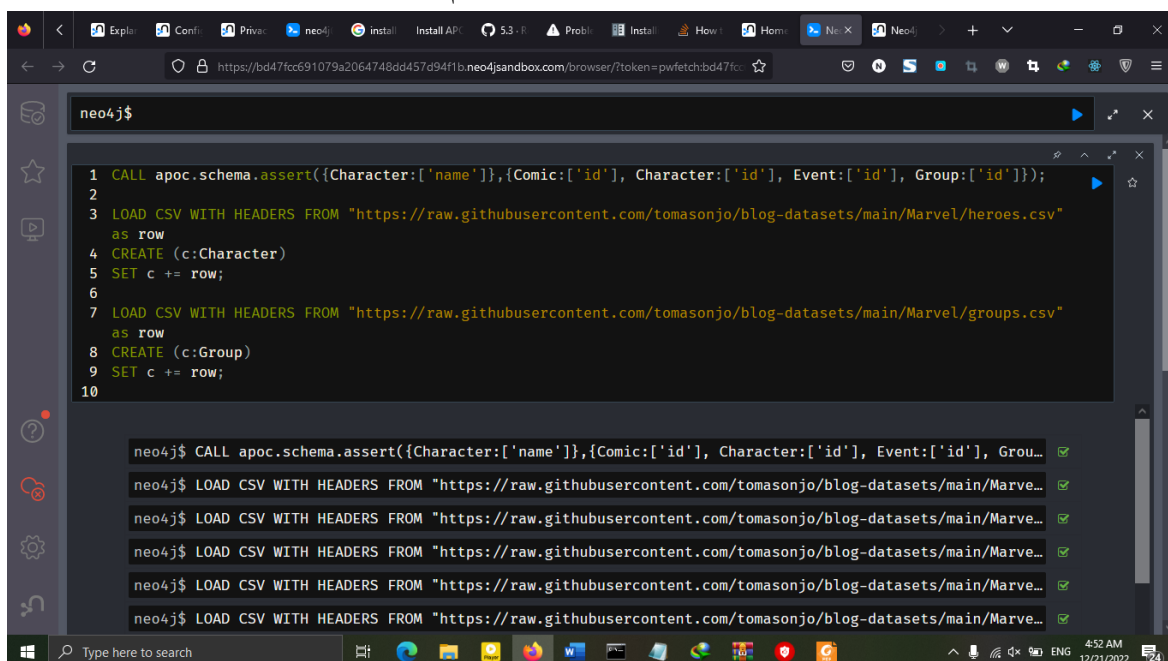
The interface also shows a 'play movie graph' button and a 'The Movie Graph' section at the bottom.

یک دور چک میکنیم که دیتاست پاک شده باشد

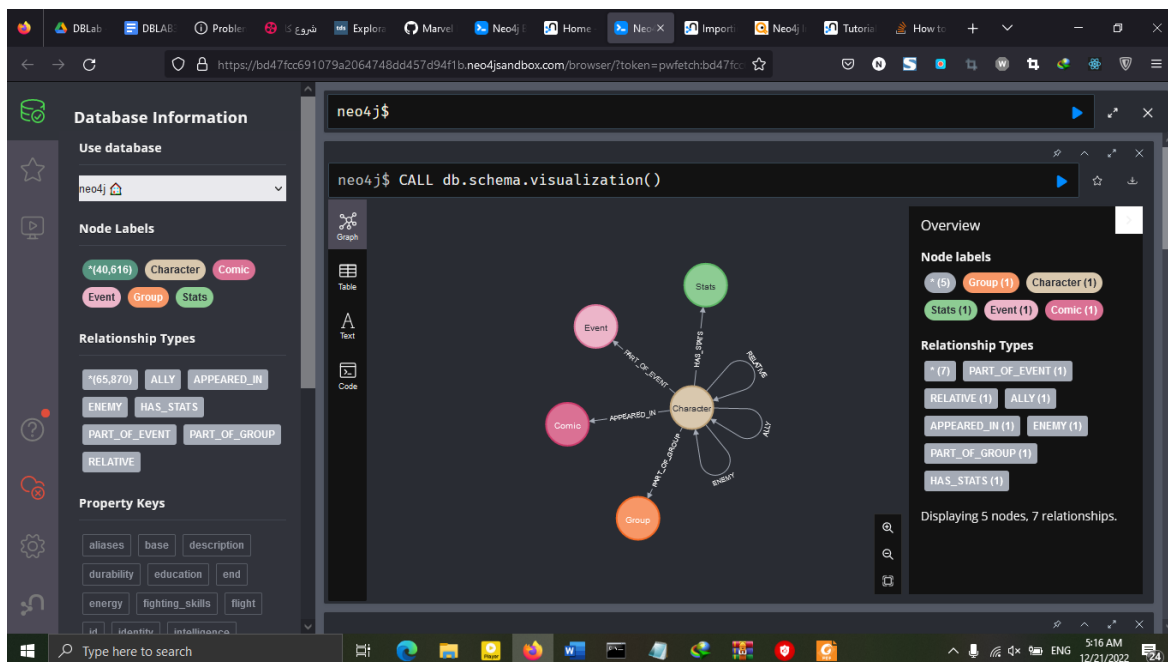


همانطور که مشاهده میکنید 0 باز میگرداند.

لود کردن دیتاست مارول. طبق دستورات موجود در مقاله دیتا را لود میکنیم.

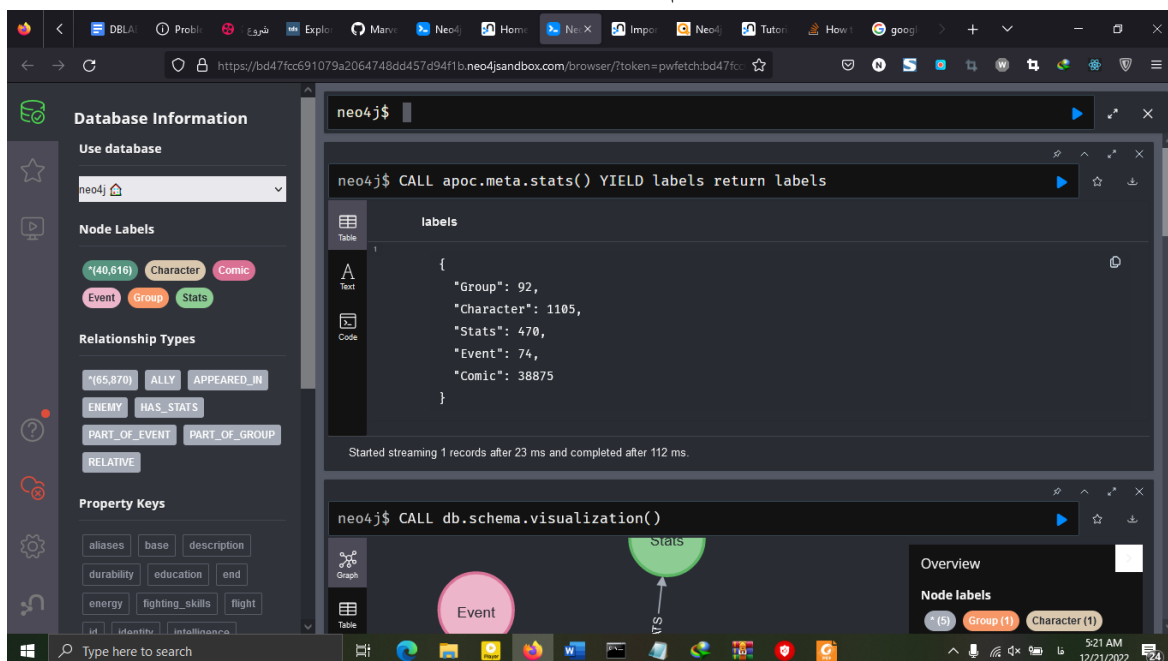


## نمایش شمای گراف



با دستور call یک procedure را صدا می‌کنیم

در مرکز characters وجود دارد که می‌تواند جزئی از یک event باشد یا در چندین کامیک ظاهر شود، یا به یک گروه تعلق داشته باشد و یا آمار داشته باشد. کاراکترها می‌توانند دشمن هم، متحد و یا فامیل باشند. برای درک بهتر سباز گراف این دستور را اجرا می‌کنیم.



در کل 1105 کاراکتر وجود دارد که در 338875 کامیک ظاهر شده است. 470 کاراکتر آمار دارند. 92 گروه 74 ایونت در گراف ذخیره شده اند.

برای تحلیل گراف تحلیل خود را با تعداد کاراکترهایی که بیشترین تکرار را در کامیک دارند شروع می‌کنیم. کوئری زیر تعداد کاراکتری که برای کامیک ظاهر شده اند را بر اساس تعداد رابطه `appeared_in` را با دستور `size` پیدا می‌کند (تعداد روابط که نود کاراکتر آن ها کاراکتر مورد نظر است) و آن ها را مرتب کرده و 5 تای اول را باز میگرداند.

The screenshot shows the Neo4j Browser interface. On the left, the 'Database Information' sidebar is visible, showing the database 'neo4j' and various node labels like Character, Comic, Event, Group, and Stats. The main area displays a Cypher query:

```
1 MATCH (c:Character)
2 RETURN c.name as character,
3        size((c)-[:APPEARED_IN]-()) as comics
4 ORDER BY comics DESC
5 LIMIT 5
```

The results are shown in a table with two columns: 'character' and 'comics'.

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

Below the table, it says: 'Started streaming 5 records after 2 ms and completed after 26 ms.'

با استفاده از کوئری زیر رویدادهایی را پیدا میکنیم که بیشترین کاراکتر را دارند.

The screenshot shows the Neo4j Browser interface. On the left, the 'Database Information' sidebar is visible. The main area displays a Cypher query:

```
1 MATCH (e:Event)
2 RETURN e.title as event,
3        size((e)-[:PART_OF_EVENT]-()) as count_of_heroes,
4        e.start as start,
5        e.end as end,
6        e.description as description
7 ORDER BY count_of_heroes DESC
8 LIMIT 5
```

The results are shown in a table with five columns: 'event', 'count\_of\_heroes', 'start', 'end', and '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 bro
"Dark Reign"	128	"2008-12-01 00:00:00"	"2009-12-31 12:59:00"	"Norman Osborn came out the herc
"Acts of Vengeance!"	93	"1989-12-10 00:00:00"	"2008-01-04 00:00:00"	"Loki sets about convincing the sup
"Secret Invasion"	89	"2008-06-02 00:00:00"	"2009-01-25 00:00:00"	"The Shape-shifting Skrulls have be
"Civil War"	86	"2006-07-01 00:00:00"	"2007-01-29 00:00:00"	"After a horrific tragedy raises ques

در این کوئری برای هر رویداد تعداد نودهایی که به آن رابطه part\_of\_event دارند را با دستور size میگیرد و براساس تعداد نودها مرتب کرده و 5 تای اولی را برمیگرداند.

## پرجمعیت ترین گروه ها

The screenshot shows the Neo4j Browser interface. On the left, the 'Database Information' sidebar is visible, showing the database name 'neo4j' and various node labels like 'Character', 'Comic', 'Event', 'Group', and 'Stats'. The main area displays a Cypher query:

```
1 MATCH (g:Group)
2 RETURN g.name as group,
3       size((g)-[:PART_OF_GROUP]-()) as members
4 ORDER BY members DESC LIMIT 5
```

The query results are shown in a table with two columns: 'group' and 'members'.

group	members
"X-Men"	41
"Avengers"	31
"Defenders"	26
"Next Avengers"	14
"Guardians of the Galaxy"	12

Below the table, it states: 'Started streaming 5 records after 24 ms and completed after 25 ms.'

در این کوئری برای هر گروه تعداد نودهایی که به آن رابطه `part_of_group` دارند را با دستور `size` میگیرد و براساس تعداد نودها مرتب کرده و 5 تای اولی را برمیگرداند. کاراکترهایی که باهم دشمن اند ولی عضو یک گروهند.

The screenshot shows the Neo4j Browser interface with a different Cypher query:

```
1 MATCH (c1:Character)-[:PART_OF_GROUP]->(g:Group)-[:PART_OF_GROUP]-(c2:Character)
2 WHERE (c1)-[:ENEMY]-(c2) and id(c1) < id(c2)
3 RETURN c1.name as character1, c2.name as character2, g.name as group
```

The query results are shown in a table with three columns: 'character1', 'character2', and 'group'.

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"

Below the table, it states: 'Started streaming 5 records after 33 ms and completed after 417 ms.'

ابتدا هر دو کاراکتر `c1`, `c2` ای را میابیم که عضو یک گروه باشند و سپس چک میکنیم که این دو یال دشمن داشته باشند و همینطور چک میکنیم که یکبار بیایند در جدول.

کاراکترهایی که از Yugoslavia هستند.

The screenshot shows the Neo4j Browser interface. On the left, the 'Database Information' sidebar is visible, showing 'Use database' set to 'neo4j', 'Node Labels' including Character, Comic, Event, Group, and Stats, 'Relationship Types' including ALLY, APPEARED\_IN, ENEMY, HAS\_STATS, PART\_OF\_EVENT, PART\_OF\_GROUP, and RELATIVE, and 'Property Keys' including aliases, base, description, durability, education, end, energy, fighting\_skills, flight, id, identity, and intelligence. The main area displays a Cypher query:

```
1 MATCH (c:Character)
2 WHERE c.place_of_origin contains "Yugoslavia"
3 RETURN c.name as character,
4        c.place_of_origin as place_of_origin,
5        c.aliases as aliases
```

The results are shown in a table:

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

Below the table, a second query is shown:

```
neo4j$ MATCH (c1:Character)-[:PART_OF_GROUP]-(g:Group)-[:PART_OF_GROUP]-(c2:C...
```

ابتدا چک میکنیم که محل تولد شامل Yugoslavia باشد سپس نامو محل تولد و لقب های آن کاراکتر را برمیگردانیم.

کاراکترهایی که PHD دارند

The screenshot shows the Neo4j Browser interface. On the left, the 'Database Information' sidebar is visible, showing 'Use database' set to 'neo4j', 'Node Labels' including Character, Comic, Event, Group, and Stats, 'Relationship Types' including ALLY, APPEARED\_IN, ENEMY, HAS\_STATS, PART\_OF\_EVENT, PART\_OF\_GROUP, and RELATIVE, and 'Property Keys' including aliases, base, description, durability, education, end, energy, fighting\_skills, flight, id, identity, and intelligence. The main area displays a Cypher query:

```
1 MATCH (c:Character)
2 WHERE c.education contains "Ph.D"
3 RETURN c.name as character, c.education as education
4 LIMIT 10
```

The results are shown in a table:

	character	education
1	"UNKNOWN ACHEBE"	"Ph.D. in Law (Yale), degrees in Psychology, Political Science and Divinity"
2	"PROFESSOR MENDEL STROMM MENDEL STROMM"	"Ph.D. in robotics"
3	"FRANKLIN HALL GRAVITON"	"Ph.D. in physics"
4	"Morbis"	"Ph.D in Biochemistry"
5	"Tony Stark"	"Ph.Ds in physics and electrical engineering"
6	"Hulk-dok"	"Ph.D in nuclear physics and two other fields"

چک میکنیم که کاراکتر شامل Ph.D باشد و نام و تحصیلات را برمیگردانیم با محدودیت 10 تا

کاراکترهایی که بیشترین رابطه را با بقیه دارند.

The screenshot shows the Neo4j Browser interface. On the left, the 'Database Information' panel is visible, showing the database 'neo4j' and various node labels and relationship types. The main area displays a Cypher query:

```
1 MATCH (c:Character)
2 RETURN c.name as name,
3        size((c)-[:ALLY]→()) as allies,
4        size((c)-[:ENEMY]→()) as enemies,
5        size((c)-[:RELATIVE]→()) as relative
6 ORDER BY allies + enemies + relative DESC
7 LIMIT 5
```

The results are shown in a table with 4 columns: name, allies, enemies, and relative. The data is as follows:

name	allies	enemies	relative
"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
"Kamak"	6	2	17

تعداد نودها را براساس تعداد یال های Ally, enemy, relative که با size به دست آوردیم میگیریم و براساس مجموعشان مرتب میکنیم و 5 تای اول را برمیداریم.

گراف ارتباط خانواده Triton با یکدیگر

The screenshot shows the Neo4j Browser interface with a graph visualization. The Cypher query is:

```
1 MATCH p=(c:Character{name:"Triton"})
2 CALL apoc.path.subgraphAll(id(c), {relationshipFilter:"RELATIVE"})
3 YIELD nodes, relationships
4 RETURN nodes, relationships
```

The graph shows a central node 'Triton' connected to many other nodes, representing the family structure. The 'Overview' panel on the right shows 24 node labels and 100 relationships.

با استفاده از procedure subgraphAll گرافی با تمام روابط با کاراکتر Triton را میسازیم. این تابع 2 ارگومان میگیرد اولی ایدی یک نود و دومی رابطه خواسته شده است. این تابع تمام نودهایی که با آن نود اولیه رابطه خواسته شده را دارند برمیگرداند.

## مشکلات و توضیحات تکمیلی

---

برای حل کانفیگ کتابخانه apoc خیلی به مشکل برخورددم.



## آنچه آموختم / پیشنهادات

---

در این دستور کار به noe4j آشنا شدم که بسیار جالب بود و استفاده از داکيومنت خیلی خوب بود.