

Neo4j

1. `match (n {lastName:«Smith«}) return n;`

The screenshot shows the Neo4j Browser interface. On the left, the 'Database Information' sidebar is visible, listing 'Node Labels' (Bill, Body, Committee, Congress, District, Legislator, Party, State, Subject) and 'Relationship Types' (DEALS_WITH, ELECTED_TO, IS_MEMBER_OF, PROPOSED_DURING, REFERRED_TO, REPRESENTS, SERVES_ON, SPONSORED_BY, VOTED_ON). The main area displays the results of the query `$ match (n {lastName:"Smith"}) return n;`. The results are shown in a graph view with 5 nodes, all labeled 'Smith'. The status bar at the bottom indicates 'Displaying 5 nodes, 0 relationships.'

2. `match (z:Legislator), p=(bill)-[r:SPONSORED_BY]->(l:Legislator) where z.state="CA" AND z.birthday>"1952-02-09" return p limit 25;`

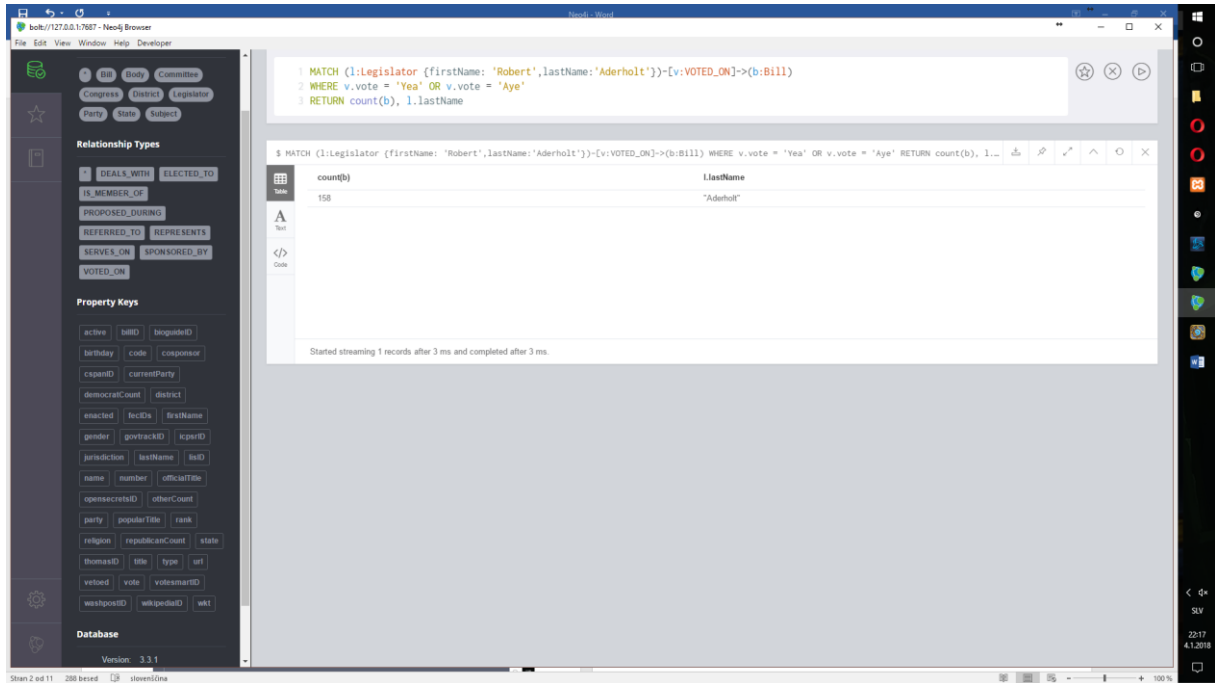
The screenshot shows the Neo4j Browser interface with a more complex query: `$ match (z:Legislator), p=(bill)-[r:SPONSORED_BY]->(l:Legislator) where z.state="CA" AND z.birthday>"1952-02-09" return p limit 25;`. The results are displayed in a graph view, showing a central node labeled 'Drazen' connected to 25 other nodes, all labeled 'False'. The status bar at the bottom indicates 'Displaying 26 nodes, 25 relationships.'

3)

```
MATCH (l:Legislator {firstName: 'Robert',lastName:'Aderholt'})-[v:VOTED_ON]->(b:Bill)
```

```
WHERE v.vote = 'Yea' OR v.vote = 'Aye'
```

```
RETURN count(b), l.lastName
```



4)

```
MATCH (c:Congress {number:"114"})
```

```
WITH c, size((l)-[:PROPOSED_DURING]->(c)) as totalProposed
```

```
MATCH (b:Bill {active:"True"})-[:PROPOSED_DURING]->(c)
```

```
RETURN totalProposed, count(b) as activeProposed
```

The screenshot shows the Neo4j Desktop interface. On the left is a sidebar with navigation options like 'Graph', 'Data', 'Tools', and 'Settings'. The main area displays a Cypher query and its results.

```

1 MATCH (c:Congress {number:"114"})
2 WITH c, size()-[:PROPOSED_DURING]->(c) as totalProposed
3 MATCH (b:Bill {active:"True"})-[:PROPOSED_DURING]->(c)
4 RETURN totalProposed, count(b) as activeProposed

```

The results table shows two columns: `totalProposed` and `activeProposed`. The first row contains the values 10223 and 2105.

totalProposed	activeProposed
10223	2105

Below the table, a status message reads: "Started streaming 1 records after 43 ms and completed after 43 ms."

5)

MATCH (p:Legislator)-[r:REPRESENTS]->(z:State)

WITH z AS drzave, sum(size((p{currentParty:"Democrat"})-[r]->(z))) AS sestevek_dem,
sum(size((p{currentParty:"Republican"})-[r]->(z))) AS sestevek_rep

WHERE sestevek_dem>sestevek_rep

RETURN drzave.code

The screenshot shows the Neo4j Web Interface with a Cypher query and its results.

Database Information:

- Node Labels:** Bill, Body, Committee, Congress, District, Legislator, Party, State, Subject
- Relationship Types:** DEALS_WITH, ELECTED_TO, IS_MEMBER_OF, PROPOSED_DURING, REFERRED_TO, REPRESENTS, SERVES_ON, SPONSORED_BY, VOTED_ON
- Property Keys:** active, billID, bioguideID, birthday, code, cosponsor, cspanID, currentParty, democratCount, district, enacted, fecIDs, firstName, gender, govtrackID, icpsid, jurisdiction, lastName, lastID, name, number, officialTitle, openSecretsID, otherCount, party, popularTitle, rank, religion, republicanCount, state, thomasID, title, type, url, voted, vote, voteformID

Cypher Query:

```
1 MATCH (p:Legislator)-[r:REPRESENTS]->(z:State)
2 WITH z AS drzave, sum(size((p(currentParty:"Democrat"))-[r]->(z))) AS sestevek_dem, sum(size((p(currentParty:"Republican"))-[r]->(z))) AS sestevek_rep
3 WHERE sestevek_dem>sestevek_rep
4 RETURN drzave.code
```

Results:

drzave.code
"HI"
"MA"
"RI"
"NY"
"MP"
"OR"
"IL"
"MN"
"NH"
"WA"
"NM"
"CT"
"CA"
"MD"
"VT"
"DC"

Started streaming 20 records after 31 ms and completed after 40 ms

6.

MATCH (s:State {code: "OH"})<-[r:REPRESENTS]-(l:Legislator)

MATCH (l)-[r:IS_MEMBER_OF]->(p:Party)

WITH p.name as party, count(r) as num

RETURN party, num;

The screenshot shows a web browser with a Cypher query editor and results. The query is:

```
1 MATCH (s:State {code: "OH"})<-[:REPRESENTS]-(:Legislator)
2 MATCH (l)-[:IS_MEMBER_OF]->(p:Party)
3 WITH p.name as party, count(r) as num
4 RETURN party, num;
```

The results are displayed in a table with two columns: party and num.

party	num
"Democrat"	5
"Republican"	13

Below the table, it says: "Started streaming 2 records after 1 ms and completed after 6 ms."

7.)

The screenshot shows a web browser with a Cypher query and its results as a graph. The query is:

```
$ MATCH (l:Legislator)-[:ELECTED_TO]->(b:Body{type:"Senate"}) MATCH (l)-[:SERVES_ON]->(c:Committee) MATCH (l)-[:REPRESENTS]->(st:State{code:"FL"}) RETURN l, c, b, st;
```

The results are displayed as a graph with 12 nodes and 13 relationships. The nodes are labeled: "Henderson", "Pitt", "Ridley", "Senate General Counsel", "Senate Special Counsel", "Senate Counsel", "Senate Counsel", "Senate Counsel", "Senate Counsel", "Senate Counsel", "Senate Counsel", "Senate Counsel", "Senate Counsel". The relationships are labeled: "ELECTED_TO", "SERVES_ON", "REPRESENTS".

At the bottom, it says: "Displaying 12 nodes, 13 relationships."

```
MATCH (l:Legislator)-[e:ELECTED_TO]->(b:Body{type:"Senate"})
```

```
MATCH (l)-[s:SERVES_ON]->(c:Committee)
```

```
MATCH (l)-[r:REPRESENTS]->(st:State{code:"FL"})
```

```
RETURN l, c, b, st;
```

8.)

```
MATCH (l:Legislator)-[se:SERVES_ON]->(c:Committee)
```

```
MATCH (l)-[r:REPRESENTS]->(st:State{code:"FL"})
```

```
MATCH (bi:Bill)-[d:DEALS_WITH]->(s:Subject)
```

```
RETURN s
```

```
LIMIT 25;
```

The screenshot displays the Neo4j browser interface. On the left, the 'Database Information' sidebar shows 'Node Labels' (Bill, Body, Committee, Congress, District, Legislator, Party, State, Subject) and 'Relationship Types' (DEALS_WITH, ELECTED_TO, IS_MEMBER_OF, PROPOSED_DURING, REFERRED_TO, REPRESENTS, SERVES_ON, SPONSORED_BY, VOTED_ON). The 'Property Keys' section lists various attributes like active, billID, bioguideID, birthday, code, cosponsor, etc.

The main area shows a Cypher query in the editor:

```
1 MATCH (l:Legislator)-[se:SERVES_ON]->(c:Committee)
2 MATCH (l)-[r:REPRESENTS]->(st:State{code:"FL"})
3 MATCH (bi:Bill)-[d:DEALS_WITH]->(s:Subject)
4 RETURN s
5 LIMIT 25;
```

Below the query, the results are displayed in a table. The first column is labeled 's' and contains a JSON object representing a subject:

```
{
  "title": "Veterans' organizations and recognition"
}
```

The second column is labeled 'c' and contains a JSON object representing a committee:

```
{
  "name": "House Committee on Ways and Means",
  "type": "house",
  "url": "http://waysandmeans.house.gov/",
  "thomasID": "HGMH",
  "jurisdiction": "The Committee on Ways and Means is the oldest committee of the United States Congress, and is the chief tax-writing committee in the House of Representatives. The Committee derives a large share of its jurisdiction from Article I, Section VII of the U.S. Constitution which declares, \"All Bills for raising Revenue shall originate in"
}
```

At the bottom, a status message indicates: 'Started streaming 25 records after 1 ms and completed after 5 ms.'

GAME OF THRONES

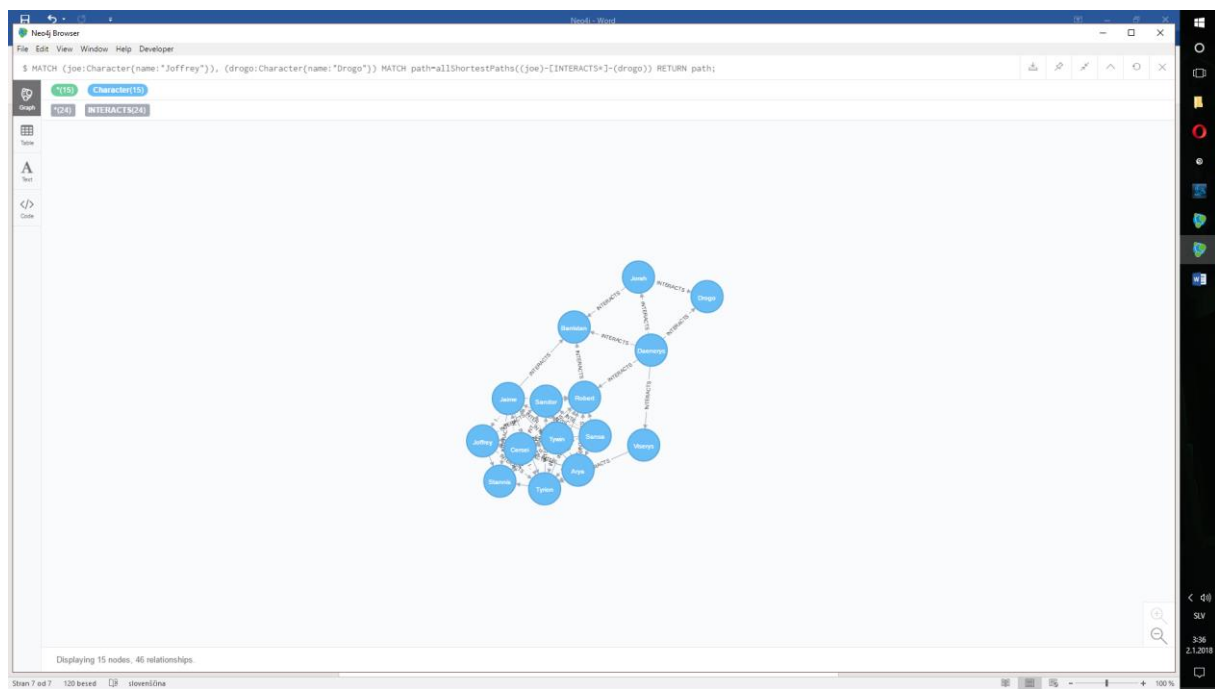
```
1) MATCH (c:Character)-[:INTERACTS]->>()  
   WITH c, count(*) AS num  
   RETURN min(num) AS min, max(num) AS max, avg(num) AS  
          avg_characters, stdev(num) AS stdev
```

\$ MATCH (c:Character)-[:INTERACTS]->>() WITH c, count(*) AS num RETURN min(num) AS min, max(num) AS max, avg(num) AS avg_characters, stdev(num) AS ...

min	max	avg_characters	stdev
1	24	4.957746478873241	6.227672391875085

Started streaming 1 records after 42 ms and completed after 42 ms.

```
2) MATCH (joe:Character{name:"Joffrey"}), (drogo:Character{name:"Drogo"})  
   MATCH path=allShortestPaths((joe)-[:INTERACTS*]-(drogo))  
   RETURN path;
```



```
3. MATCH (a:Character), (b:Character) WHERE id(a) > id(b)
MATCH p=shortestPath((a)-[:INTERACTS*]-(b))
RETURN length(p) AS len, extract(x IN nodes(p) | x.name) AS path
ORDER BY len DESC LIMIT 4
```


The screenshot shows the Neo4j Browser interface. The top panel displays a Cypher query: `$ MATCH (a:Character), (b:Character) WHERE id(a) > id(b) MATCH p=shortestPath((a)-[:INTERACTS*]-(b)) RETURN length(p) AS len, extract(x IN nodes(p) | x.name) AS path ORDER BY len DESC LIMIT 4`. The results are shown in a table with two columns: **len** and **path**. The table contains four rows of data, each representing a path of length 5 between two characters. Below the table, a status message indicates: "Started streaming 4 records after 2465 ms and completed after 2479 ms." The bottom panel shows the Neo4j logo and three main sections: "Learn about Neo4j", "Jump into code", and "Monitor the system".

len	path
5	["Ilyrio", "Belwas", "Daenerys", "Robert", "Tywin", "Oberyn", "Amory"]
5	["Ilyrio", "Belwas", "Daenerys", "Robert", "Sansa", "Bran", "Jojen"]
5	["Ilyrio", "Belwas", "Daenerys", "Robert", "Stannis", "Davos", "Shireen"]
5	["Ilyrio", "Belwas", "Daenerys", "Robert", "Sansa", "Bran", "Lewell"]

4.

`MATCH (rob:Character {name: "Robert Arryn"}), (aemon:Character {name: "Aemon"})`

`MATCH p=allShortestPaths((rob)-[:INTERACTS*]-(aemon))`

`RETURN p`

The screenshot shows the Neo4j Browser interface. The top panel displays a Cypher query: `$ MATCH (rob:Character {name: "Robert Arryn"}), (aemon:Character {name: "Aemon"}) MATCH p=allShortestPaths((rob)-[:INTERACTS*]-(aemon)) RETURN p`. The results are shown in a graph visualization. The graph displays a path of length 5 between two characters, Robert Arryn and Aemon. The path is highlighted in blue. The graph also shows other characters and their relationships. The bottom panel shows the graph structure with nodes and edges.

5.

The screenshot shows the Neo4j Browser interface. On the left, the 'Database Information' sidebar displays details for the 'graph.db' database, including version 3.3.1 and size 162.91 KiB. The main area shows two Cypher queries and their results.

Query 1:

```
$ MATCH (cx:Character) RETURN cx.name AS character, size( (cx)-[:INTERACTS]-() ) AS degreemax ORDER BY degreemax ASC LIMIT 5
```

character	degreemax
"Shireen"	1
"Walton"	1
"Amory"	1
"Karl"	1
"Blythe"	1

Started streaming 5 records after 7 ms and completed after 7 ms.

Query 2:

```
$ MATCH (c:Character) RETURN c.name AS character, size( (c)-[:INTERACTS]-() ) AS degree ORDER BY degree DESC LIMIT 5
```

character	degree
"Tyron"	36
"Sansa"	26
"Jae"	26
"Robb"	25
"Jaime"	24

Started streaming 5 records after 3 ms and completed after 3 ms.

MATCH (cx:Character)

RETURN cx.name AS character, size((cx)-[:INTERACTS]-()) AS degreemax ORDER BY degreemax ASC LIMIT 5

MATCH (c:Character)

RETURN c.name AS character, size((c)-[:INTERACTS]-()) AS degree ORDER BY degree DESC LIMIT 5

6.

LAHKO TUDI MATCH (C:CHARACTER)-[R:INTERACTS]-(), RAZLIČNI REZULTATI!

```

MATCH (c:Character)-[r:INTERACTS]->()
RETURN DISTINCT(c.name) AS peder, AVG(r.weight) AS teza
ORDER BY teza ASC LIMIT 5;

```

The screenshot shows the Neo4j Desktop interface with two queries and their results.

Query 1:

```
$ MATCH (c:Character)-[r:INTERACTS]->() RETURN DISTINCT(c.name) AS uni, AVG(r.weight) AS teza ORDER BY teza DESC LIMIT 5;
```

uni	teza
"Hodor"	38
"Bran"	31.444444444444443
"Jaeger"	22
"Samwell"	19.000000000000004
"Jon"	18.904761904761905

Started streaming 5 records after 9 ms and completed after 9 ms.

Query 2:

```
$ MATCH (c:Character)-[r:INTERACTS]->() RETURN DISTINCT(c.name) AS uni, AVG(r.weight) AS teza ORDER BY teza ASC LIMIT 5;
```

uni	teza
"Balon"	4
"Lewie"	4
"Varrys"	4
"Podrick"	4
"Robert Arwyn"	4

Started streaming 5 records after 7 ms and completed after 7 ms.

```

MATCH (c:Character)-[r:INTERACTS]->()
RETURN DISTINCT(c.name) AS uni, AVG(r.weight) AS teza
ORDER BY teza DESC LIMIT 5;

```

BETWEEN

```

MATCH (c:Character)
WITH collect(c) AS characters
CALL apoc.algo.betweenness(['INTERACTS'], characters, 'BOTH') YIELD node, score
SET node.betweenness = score
RETURN node.name AS name, score ORDER BY score DESC

```

The screenshot shows the Neo4j Desktop interface. On the left, the 'Database Information' sidebar displays details for the 'graph.db' database, including its version (3.3.1), edition (Enterprise), and size (196.85 KiB). The main workspace contains a Cypher query editor with the following code:

```
1 MATCH (c:Character)
2 WITH collect(c) AS characters
3 CALL apoc.algo.betweenness(['INTERACTS'], characters, 'BOTH') YIELD node, score
4 SET node.betweenness = score
5 RETURN node.name AS name, score ORDER BY score DESC
```

Below the editor, the query results are displayed in a table with two columns: 'name' and 'score'. The results are ordered by score in descending order.

name	score
"Jon"	1279.7533534055322
"Robert"	1165.6025171231624
"Tyrion"	1101.3849724234349
"Daenerys"	874.8372110588583
"Robb"	796.5572832464792
"Sansa"	785.1985623519137
"Stannis"	571.5247305125714
"Jaime"	556.185222089822
"Arya"	443.81358430043337
"Tywin"	364.7212195628086
"Bran"	350.6873757604443
"Davos"	312
"Catelyn"	272.1612369898086
"Bartolstian"	223.23482676662792
"Samwell"	179.59717813179518
"Cersei"	148.84389896741845

At the bottom of the results pane, a status message reads: 'Set 107 properties, started streaming 107 records after 640 ms and completed after 643 ms.'

CLOSENESS

MATCH (c:Character)

WITH collect(c) AS characters

CALL apoc.algo.closeness(['INTERACTS'], characters, 'BOTH') YIELD node, score

RETURN node.name AS name, score ORDER BY score DESC

The screenshot shows the Neo4j Desktop interface with the same sidebar as the first image. The main workspace displays a new Cypher query:

```
1 MATCH (c:Character)
2 WITH collect(c) AS characters
3 CALL apoc.algo.closeness(['INTERACTS'], characters, 'BOTH') YIELD node, score
4 RETURN node.name AS name, score ORDER BY score DESC
```

The query results are shown in a table with 'name' and 'score' columns, ordered by score in descending order.

name	score
"Tyrion"	0.004830917874396135
"Sansa"	0.004807692307692308
"Robert"	0.0047169811320754715
"Robb"	0.004608294930875576
"Arya"	0.0045871559633027525
"Jaime"	0.004524886877828055
"Jon"	0.004524886877828055
"Stannis"	0.004524886877828055
"Tywin"	0.004424778761061947
"Eddard"	0.004347826080956522
"Cersei"	0.0041841004184100415
"Catelyn"	0.004166666666666667
"Joffrey"	0.004149377593369996
"Bran"	0.003968253968253968
"Sandor"	0.003937007874015748
"Randy"	0.00392156862745098

A status message at the bottom of the results pane states: 'Started streaming 107 records after 153 ms and completed after 159 ms.'