

Practical 8

18-3-25

Neo4j

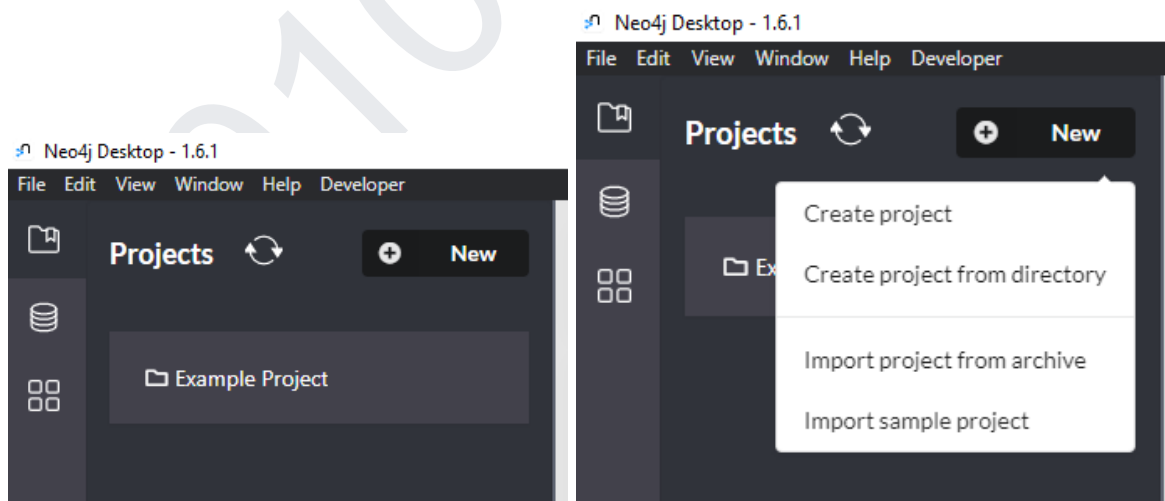
Write-up: -

- Graph databases
- Graph databases vs Relational DBMS
- Neo4j
- Cypher query language
- Any 5 CQL constructs

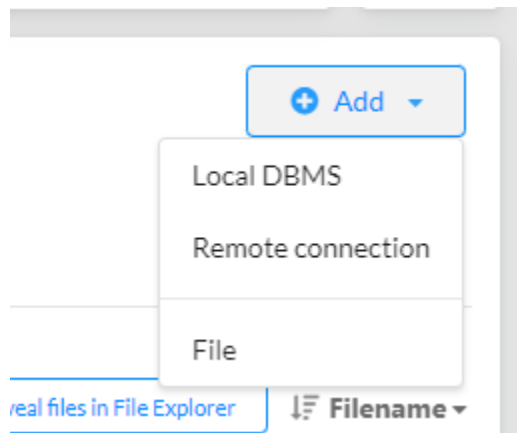
Basic Commands (Tutorials Point)

Starting Server

Create new project



Add DBMS (local DBMS)



Project

+ Add

Name

Graph DBMS

Password

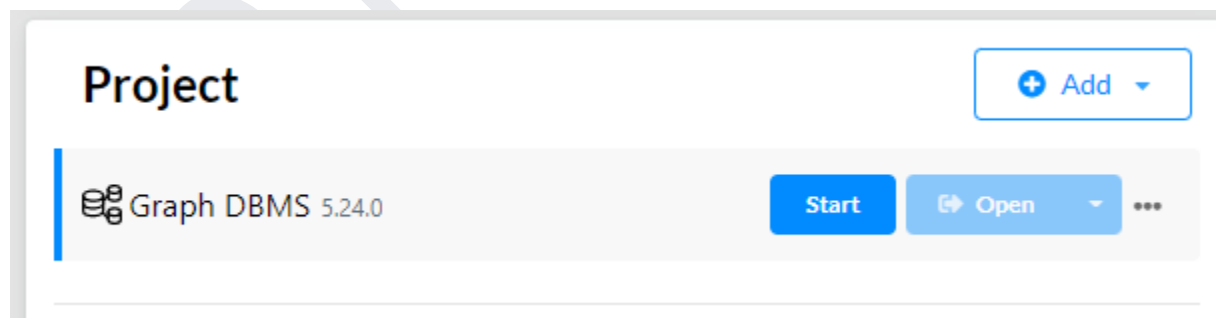
••••

Version

5.24.0

✕ Cancel ✓ Create

Start DBMS



Create Database

Project

+ Add

Graph DBMS 5.24.0

ACTIVE

Stop



Open



system

neo4j (default)

+ Create database

Refresh

Choose a name

rehmah

Cancel

Create

Open Neo4J Browser

Stop



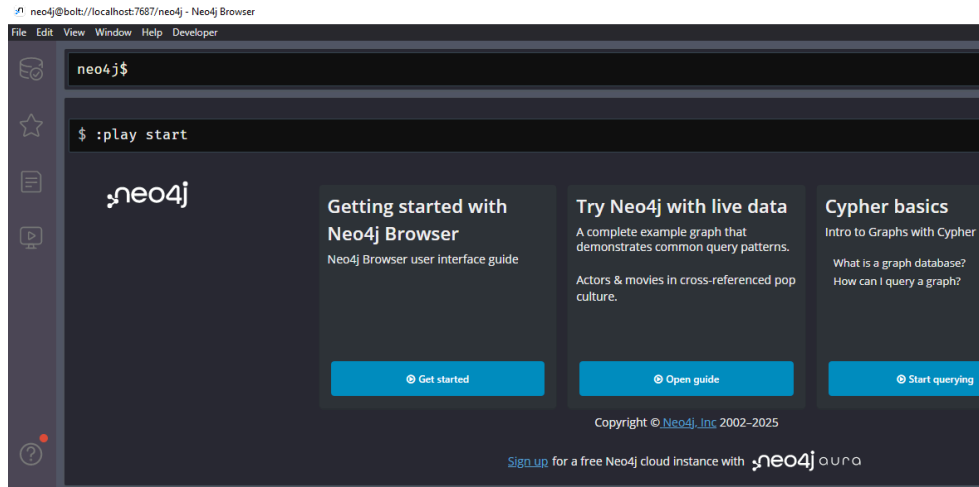
Open



Neo4j Browser

Neo4j Bloom

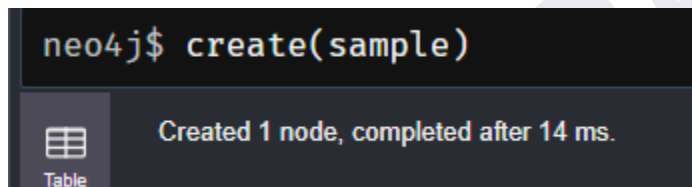
Charts



Creating a Node

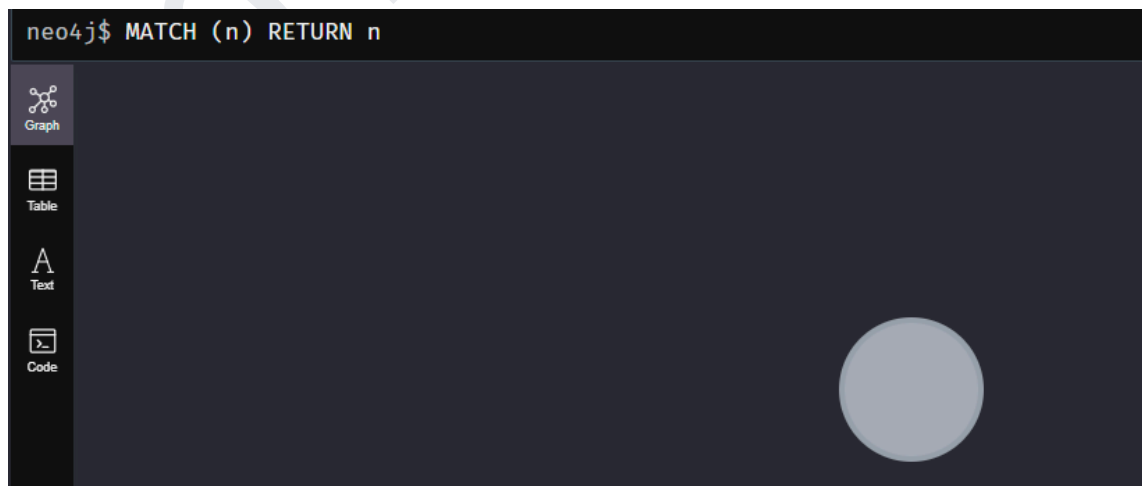
Creating a Single node

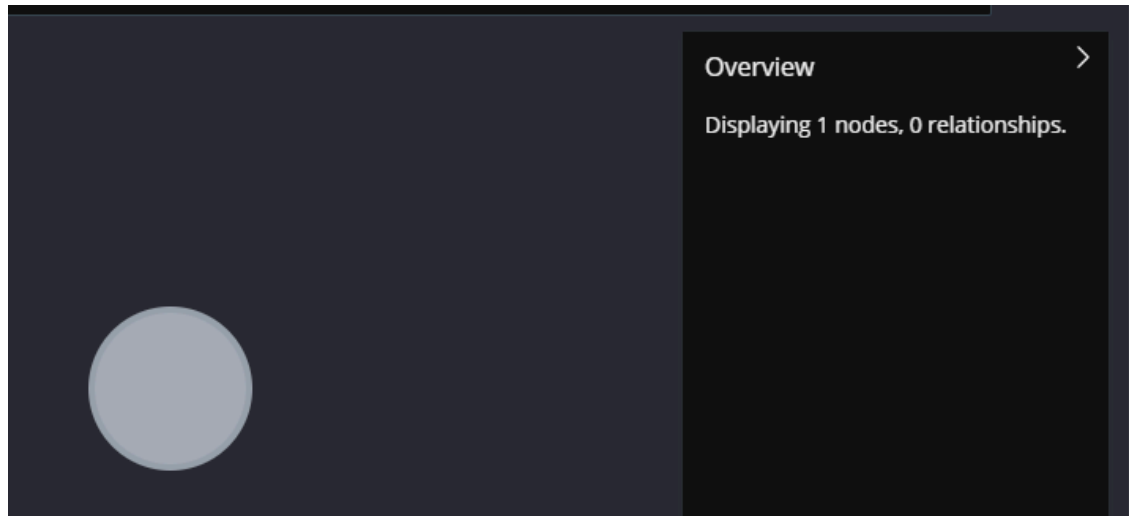
`create(sample)`



Viewing all nodes

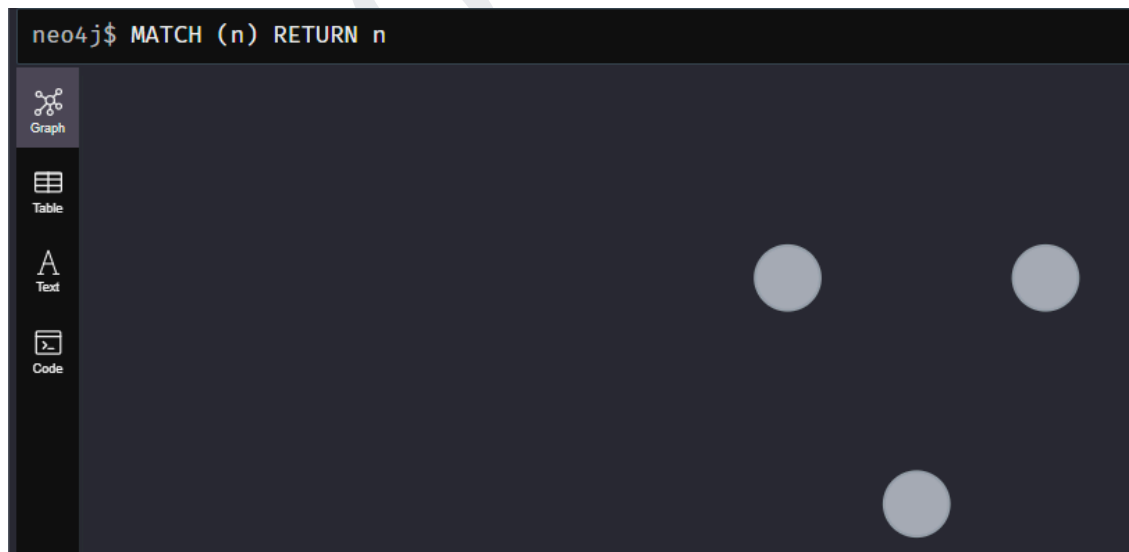
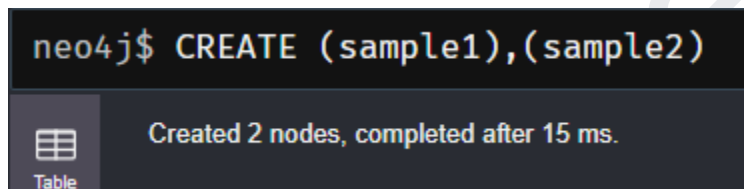
`MATCH (n) RETURN n`





Creating Multiple Nodes

CREATE (sample1),(sample2)

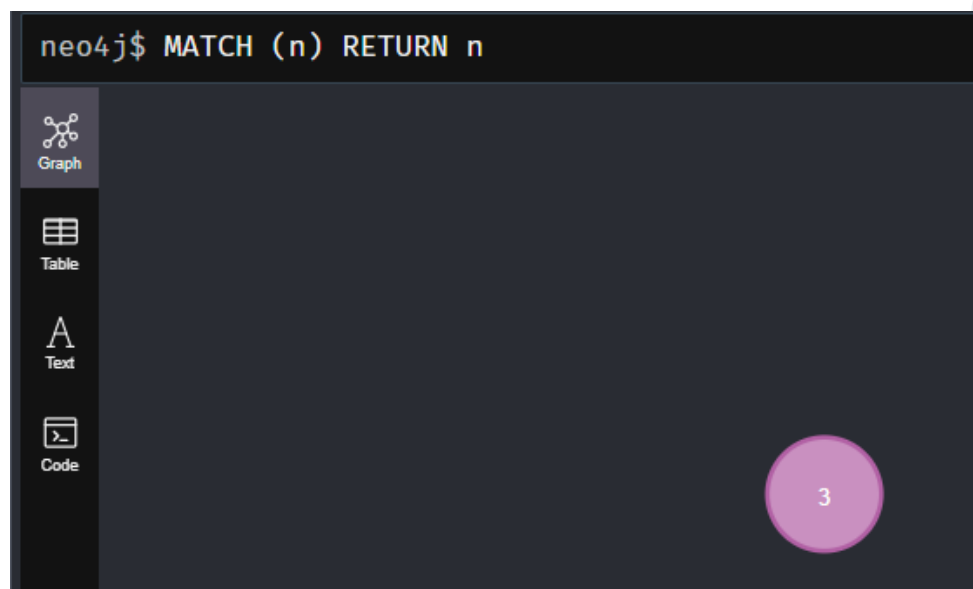


Creating a Node with a Label

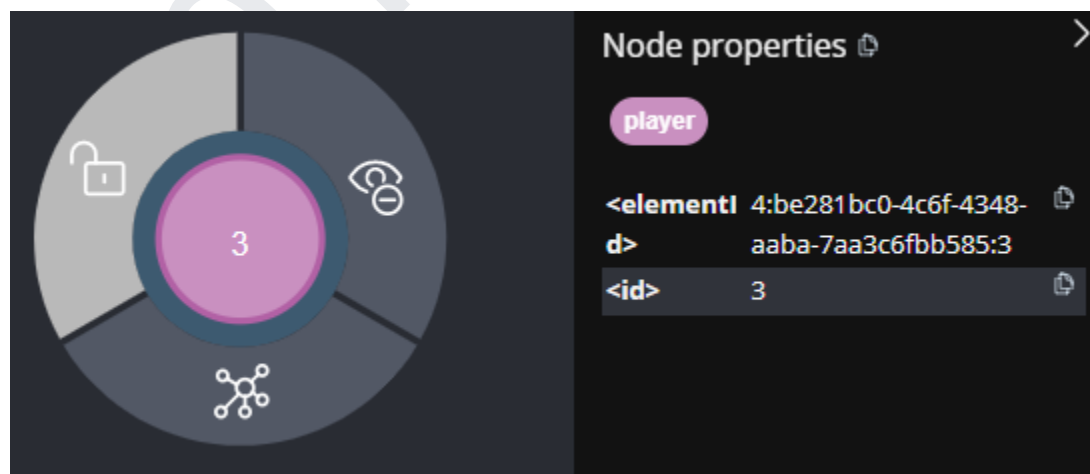
CREATE (Dhawan:player)

```
neo4j$ CREATE (Dhawan:player)
```

Added 1 label, created 1 node, completed after 32 ms.



Click on node to see label

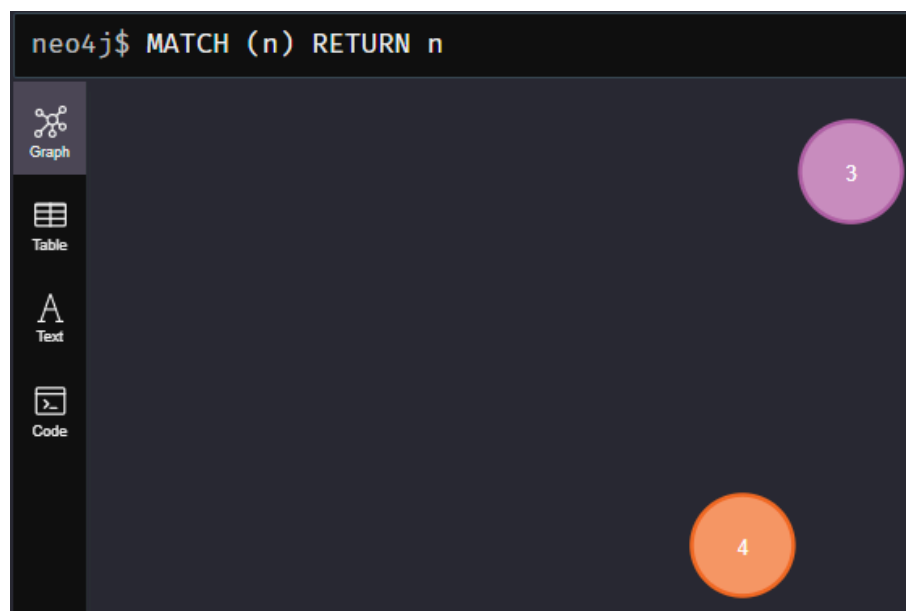


Creating a Node with Multiple Labels

CREATE (Dhawan:person:player)

```
neo4j$ CREATE (Dhawan:person:player)
```

Added 2 labels, created 1 node, completed after 16 ms.



Click on node to see labels

Node properties

person player

<elementId> 4:be281bc0-4c6f-4348-aaba-7aa3c6fbb585:4

<id> 4

Create Node with Properties

CREATE (Dhawan:player{name: "Shikar Dhawan", YOB: 1985, POB: "Delhi"})

```
neo4j$ CREATE (Dhawan:player{name: "Shikar Dhawan", YOB: 1985, POB: "Delhi"})
```

Added 1 label, created 1 node, set 3 properties, completed after 30 ms.

Match query > click on node

The image shows the Neo4j web interface. On the left, a circular graph visualization displays a single node labeled 'Shikar Dhawan' in a pink circle. The node is connected to four surrounding segments, each with an icon: a lock, a person, a network, and a database. On the right, the 'Node properties' panel is open, showing the following details:

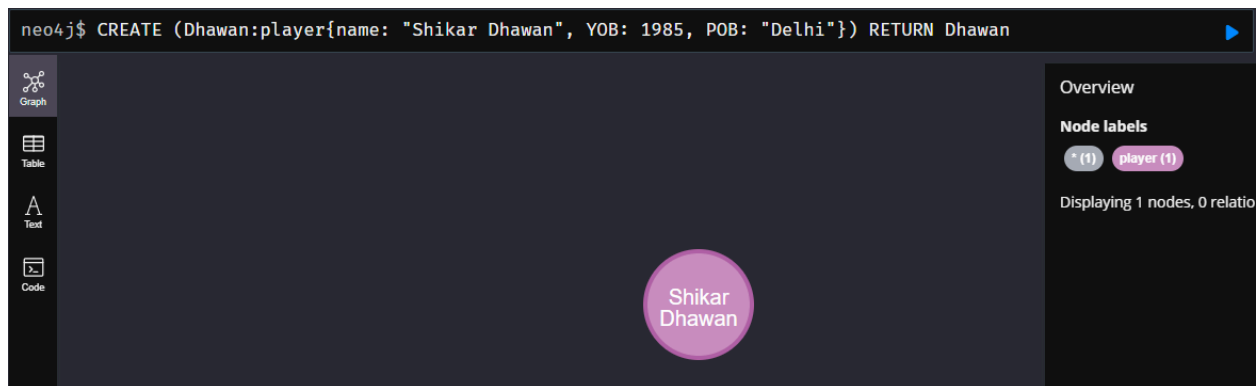
Node properties	
player	
<elementId>	4:be281bc0-4c6f-4348-aaba-7aa3c6fbb585:5
<id>	5
POB	Delhi
YOB	1985
name	Shikar Dhawan

Returning the Created Node

It will create and view the node in the same query. No need to use 'match' query.

'Return' will only show the node mentioned whereas 'match' is used to view all nodes

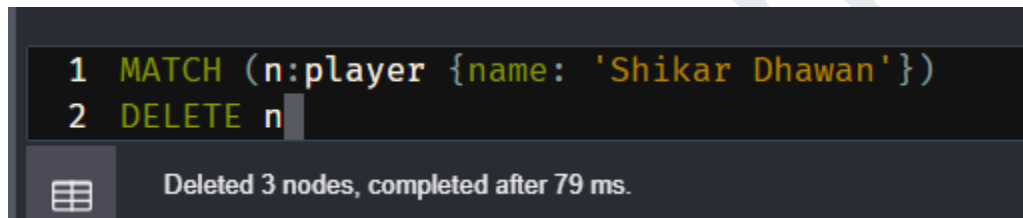
CREATE (Dhawan:player{name: "Shikar Dhawan", YOB: 1985, POB: "Delhi"}) RETURN Dhawan



Deleting a Node

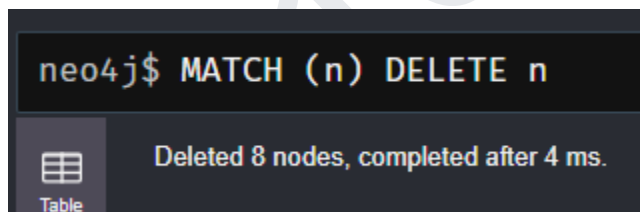
`MATCH (n:player {name: 'Shikar Dhawan'})`

`DELETE n`



Deleting all Nodes

`MATCH (n) DELETE n`



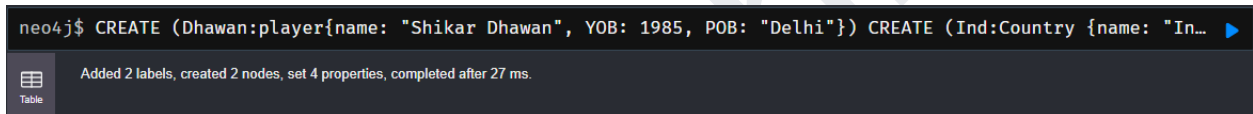
Creating Relationships

Syntax-

```
MATCH (a:LabeofNode1), (b:LabeofNode2)
  WHERE a.name = "nameofnode1" AND b.name = "nameofnode2"
CREATE (a)-[:Relation]->(b)
RETURN a,b
```

```
CREATE (Dhawan:player{name: "Shikar Dhawan", YOB: 1985, POB: "Delhi"})
```

```
CREATE (Ind:Country {name: "India"})
```



match(n) return n



```
MATCH (a:player), (b:Country)
  WHERE a.name = "Shikar Dhawan" AND b.name = "India"
CREATE (a)-[r:BATSMAN_OF]->(b)
RETURN r
```

```
1 MATCH (a:player), (b:Country)
2 WHERE a.name = "Shikar Dhawan" AND b.name = "India"
3 CREATE (a)-[r:BATSMAN_OF]->([b])
4 RETURN r
```

Table

1

```
{
  "identity": 1152921504606846977,
  "start": 1,
  "end": 2,
  "type": "BATSMAN_OF",
  "properties": {
  },
  "elementId": "5:be281bc0-4c6f-4348-aaba-7aa3c6fbb585:11529215046",
  "startNodeElementId": "4:be281bc0-4c6f-4348-aaba-7aa3c6fbb585:1",
  "endNodeElementId": "4:be281bc0-4c6f-4348-aaba-7aa3c6fbb585:2"
}
```

match(n) return n



We can't delete nodes which has relationships.

We must delete the relationship first

Then Delete the nodes

Delete a relationship


`match(n)-[r:BATSMAN_OF]->>() delete r`

```
neo4j$ match(n)-[r:BATSMAN_OF]->>() delete r
```

A screenshot of the Neo4j command-line interface showing the execution of a Cypher query to delete a relationship. The command is `match(n)-[r:BATSMAN_OF]->>() delete r`. The output shows a confirmation message: "Deleted 1 relationship, completed after 3 ms." The interface includes a sidebar with icons for Graph, Table, Text, and Code.

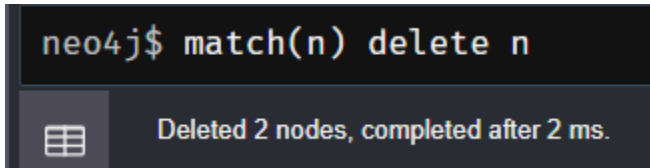
`match(n) delete n`

```
neo4j$ match(n) return n
```

A screenshot of the Neo4j web interface showing the results of a query. The query is `match(n) return n`. The results are displayed in a graph view with two nodes: "India" (a blue circle) and "Shikar Dhawan" (a pink circle). The interface includes a sidebar with icons for Graph, Table, Text, and Code.

`match(n) delete n`

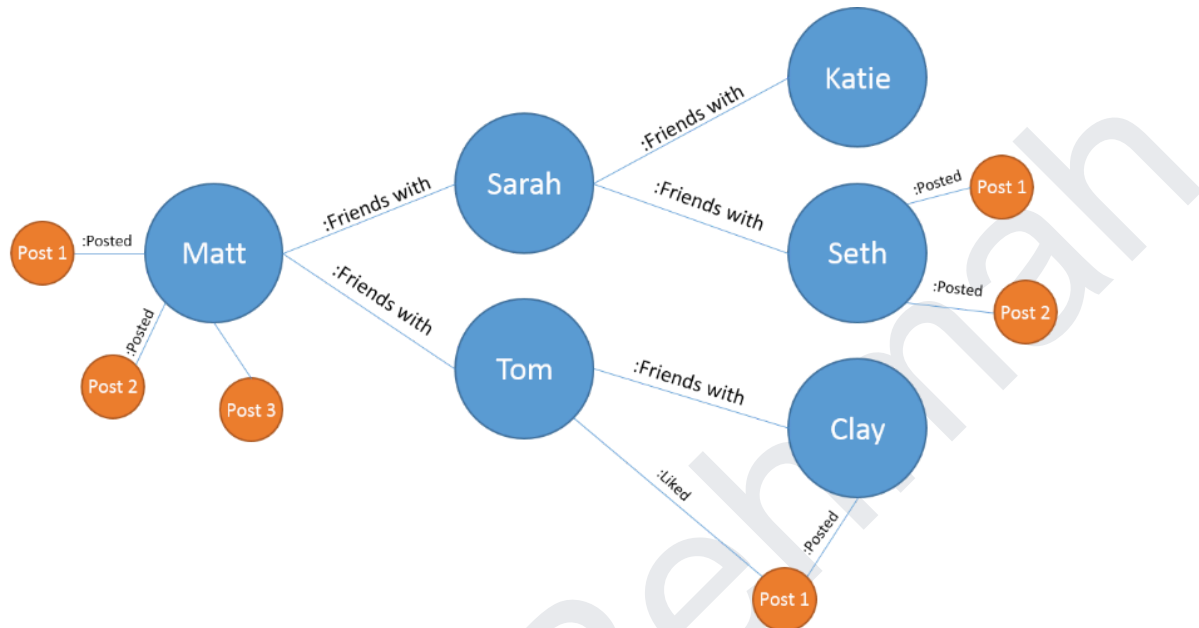
```
neo4j$ match(n) delete n
```

A screenshot of the Neo4j command-line interface showing the execution of a Cypher query to delete nodes. The command is `match(n) delete n`. The output shows a confirmation message: "Deleted 2 nodes, completed after 2 ms." The interface includes a sidebar with icons for Graph, Table, Text, and Code.

Now lets start practical on a clear graph

Even Numbers

Using the following database write the queries with Neo4j Cypher to:



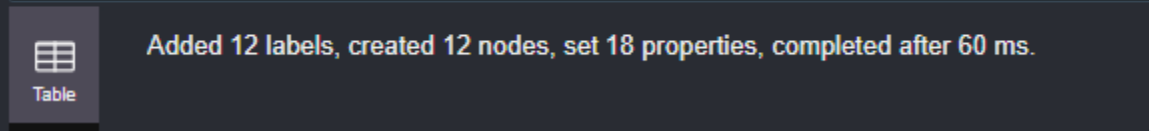
e. create the above database.

CREATE

```

(matt:Person {name: "Matt"}),
(sarah:Person {name: "Sarah"}),
(tom:Person {name: "Tom"}),
(katie:Person {name: "Katie"}),
(seth:Person {name: "Seth"}),
(clay:Person {name: "Clay"}),
(post1_matt:Post {id: "Post 1", author: "Matt"}),
(post2_matt:Post {id: "Post 2", author: "Matt"}),
(post3_matt:Post {id: "Post 3", author: "Matt"}),
(post1_seth:Post {id: "Post 1", author: "Seth"}),
(post2_seth:Post {id: "Post 2", author: "Seth"}),
(post1_clay:Post {id: "Post 1", author: "Clay"})
  
```

```
neo4j$ CREATE (matt:Person {name: "Matt"}), (sarah:Pers
```



```
MATCH (a:Person), (b:Person)
WHERE a.name = "Matt" AND b.name = "Sarah"
CREATE (a)-[r:FRIENDS_WITH]->(b)
```

Creates Relationship: Matt - Friends with - Sarah

```
MATCH (a:Person), (b:Person)
WHERE a.name = "Matt" AND b.name = "Tom"
CREATE (a)-[r:FRIENDS_WITH]->(b)
```

Creates Relationship: Matt - Friends with - Tom

```
MATCH (a:Person), (b:Person)
WHERE a.name = "Sarah" AND b.name = "Katie"
CREATE (a)-[r:FRIENDS_WITH]->(b)
```

Creates Relationship: Sarah- Friends with - Katie

```
MATCH (a:Person), (b:Person)
WHERE a.name = "Sarah" AND b.name = "Seth"
CREATE (a)-[r:FRIENDS_WITH]->(b)
```

Creates Relationship: Sarah- Friends with - Seth

```
MATCH (a:Person), (b:Person)
```

WHERE a.name = "Tom" AND b.name = "Clay"

CREATE (a)-[r:FRIENDS_WITH]->(b)

Creates Relationship: Tom- Friends with - Clay

MATCH (a:Person), (b:Post)

WHERE a.name = "Matt" AND b.author = "Matt"

CREATE (a)-[r:POSTED]->(b)

Creates Relationship:

Matt - Posted - Post1 (author is Matt)

Matt - Posted - Post2 (author is Matt)

Matt - Posted - Post3 (author is Matt)

MATCH (a:Person), (b:Post)

WHERE a.name = "Seth" AND b.author = "Seth"

CREATE (a)-[r:POSTED]->(b)

Creates Relationship:

Seth- Posted - Post1 (author is Seth)

Seth- Posted - Post2 (author is Seth)

MATCH (a:Person), (b:Post), (c:Person)

WHERE a.name = "Clay" AND b.author = "Clay" AND c.name="Tom"

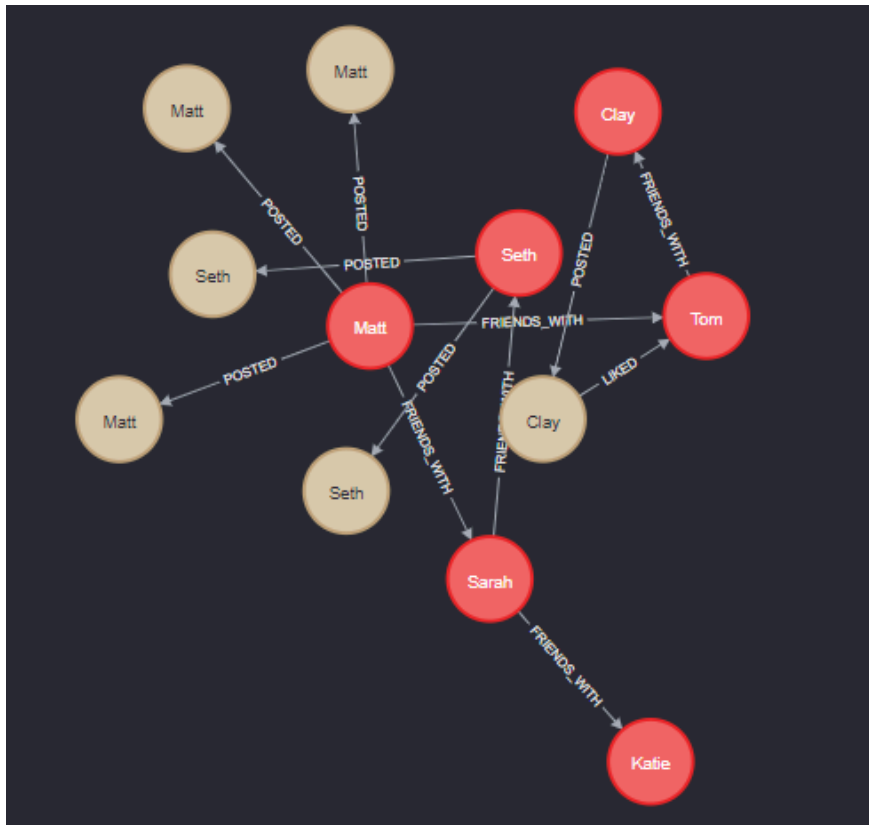
CREATE (a)-[:POSTED]->(b)-[:LIKED]-(c)

Creates Relationship:

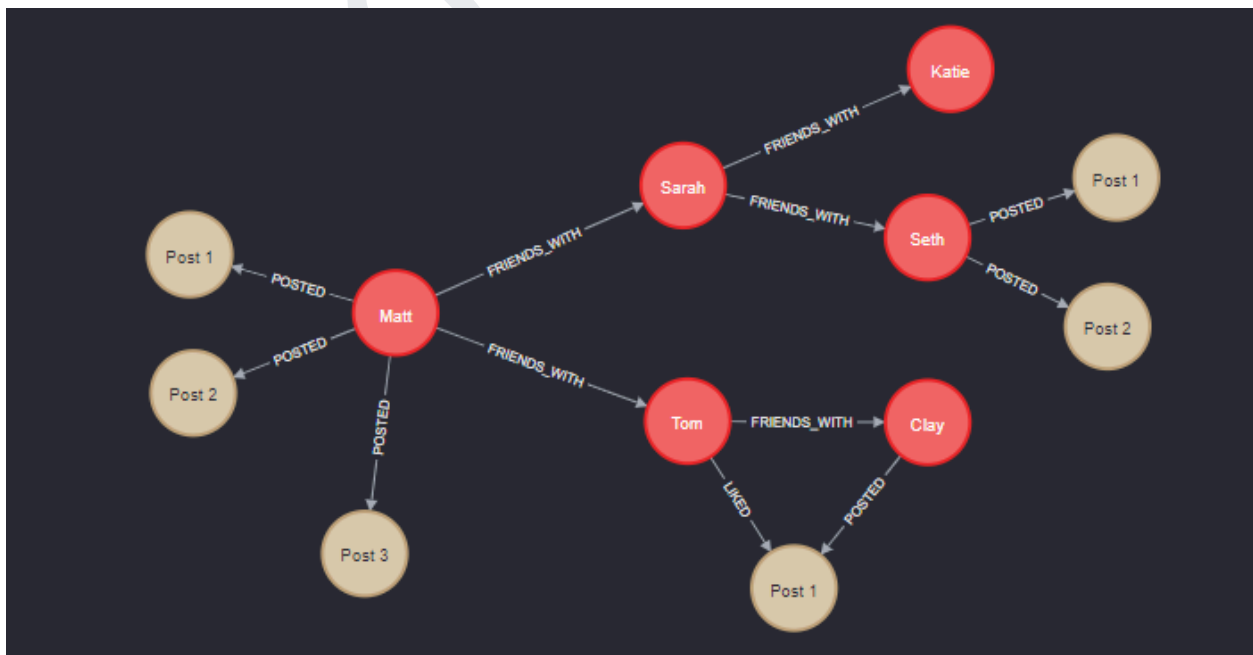
Clay- Posted - Post1 (author is Clay)

Tom- Liked - Post1 (author is clay)

match(n) return n



On Rearranging by mouse



f. know that if Katie create any posts?

```
MATCH (katie:Person {name: "Katie"})-[:POSTED]->(post:Post)
```

```
RETURN katie, post
```

```
neo4j$ MATCH (katie:Person {name: "Katie"})-[:POSTED]->(post:Post) RETURN katie, post
```

(no changes, no records)	
Table	

Katie did not create any posts

g. Know how many friends does Seth have?

```
MATCH (seth:Person {name:
```

```
"Seth"})-[:FRIENDS_WITH]-(friend:Person)
```

```
RETURN count(friend) AS numberOfFriends
```

```
neo4j$ MATCH (seth:Person {name: "Seth"})-[:FRIENDS_WITH]-(friend:Person) RETURN count(friend) AS numberOfFriends
```

	numberOfFriends
1	1

Seth has 1 friend :(

h. Know that is there a path of friendship between Matt and Clay

```
MATCH (matt:Person {name: "Matt"}), (clay:Person {name: "Clay"}),
```

```
p = shortestPath((matt)-[:FRIENDS_WITH*]-(clay))
```

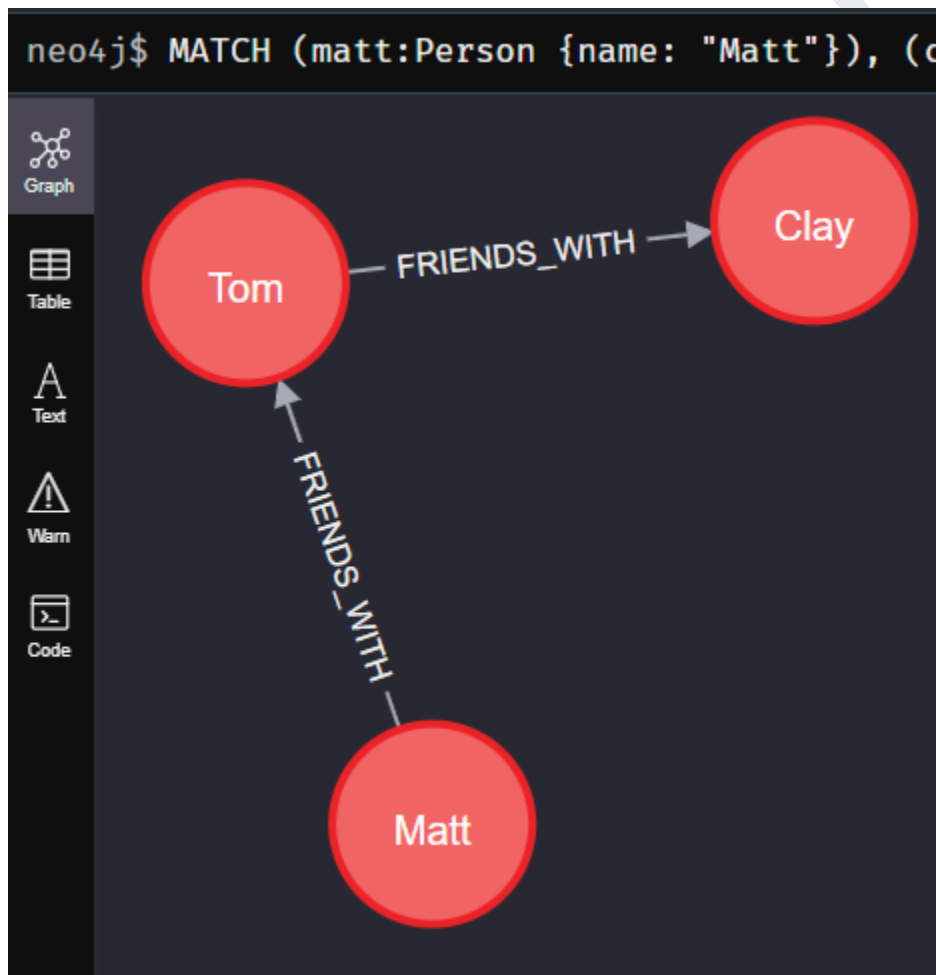
```
RETURN p
```

MATCH (**matt:Person** {**name**: "Matt"}): This finds the node representing Matt.

(**clay:Person** {**name**: "Clay"}): This finds the node representing Clay.

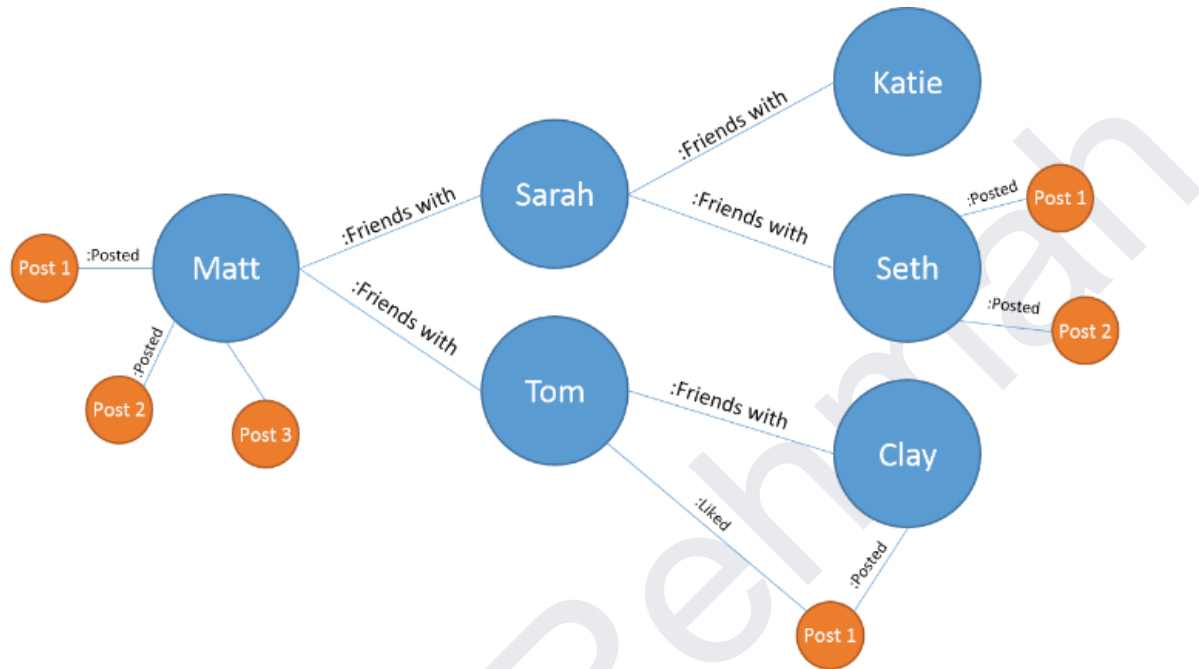
p = shortestPath((**matt**)-[:**FRIENDS_WITH***]-(**clay**)): This attempts to find the shortest path **p** between Matt and Clay, considering only **FRIENDS_WITH** relationships. The ***** allows for paths of any length.

RETURN p: This returns the shortest path found.



Odd Numbers

Using the following database write the queries with Neo4j Cypher to:



a. create the above database.

Refer Previous Question

b. Know who are Matt's friends?

```

MATCH (matt:Person {name:
"Matt"})-[:FRIENDS_WITH]-(friend:Person)
RETURN friend.name AS FriendName
  
```

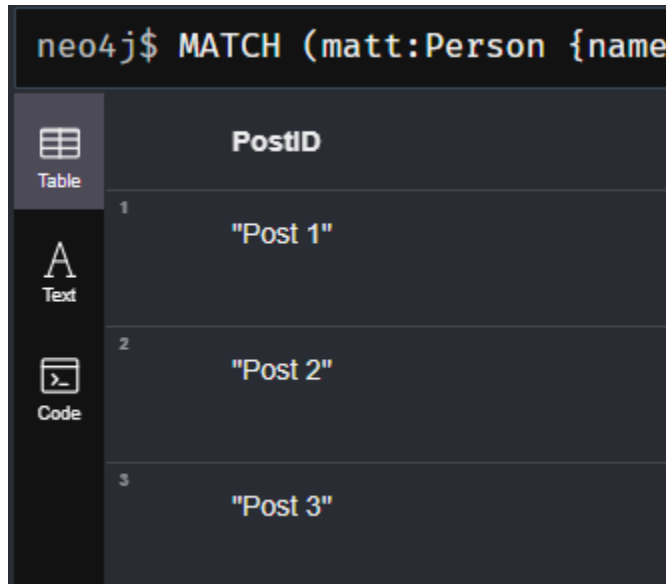
neo4j\$ MATCH (matt:Person {na

	FriendName
1	"Sarah"
2	"Tom"

c. Know what posts did Matt create?

```
MATCH (matt:Person {name: "Matt"})-[:POSTED]->(post:Post)
```

```
RETURN post.id AS PostID
```



The image shows a Neo4j query result interface. At the top, the command `neo4j$ MATCH (matt:Person {name` is visible. Below it, a table is displayed with the header **PostID**. The table contains three rows of data:

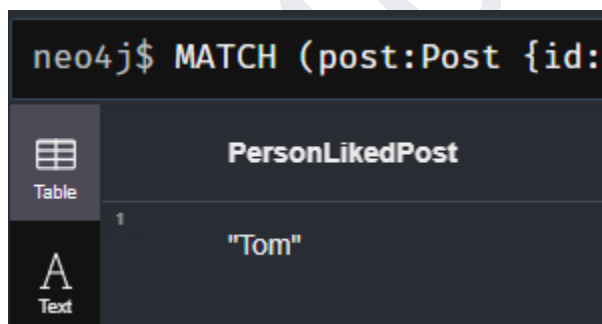
	PostID
1	"Post 1"
2	"Post 2"
3	"Post 3"

On the left side of the table, there are three icons: a table icon labeled 'Table', a text icon labeled 'Text', and a code icon labeled 'Code'.

d. Know which person liked Post 1

```
MATCH (post:Post {id: "Post 1"})<-[:LIKED]-(person:Person)
```

```
RETURN person.name AS PersonLikedPost
```



The image shows a Neo4j query result interface. At the top, the command `neo4j$ MATCH (post:Post {id:` is visible. Below it, a table is displayed with the header **PersonLikedPost**. The table contains one row of data:

	PersonLikedPost
1	"Tom"

On the left side of the table, there are two icons: a table icon labeled 'Table' and a text icon labeled 'Text'.