

Computer Science & Information Systems

Big Data Systems – Lab Sheet

Graph Database-Neo4J

Objectives

Students should be able to

- A. Gain understanding about graph databases
- B. Create a node
- C. Create relationships between nodes
- D. Retrieve nodes and relationships
- E. Update and Delete nodes and relationship

Introduction to graph databases

Graph Databases, as the name suggests, organize data in the form of a graph, based on the mathematical principle of graph theory. Fundamentally, we can consider a graph as a collection of nodes and edges. Nodes typically represent entities, edges are used to represent the relationships between those entities. What makes this useful to us in terms of databases is that nodes can hold data, describing the entity, but edges can also hold data, describing the nature and detail of that relationship. That data may be as simple as the type of relationship, or much more detailed. The sample movie graph is shown in the figure 1.

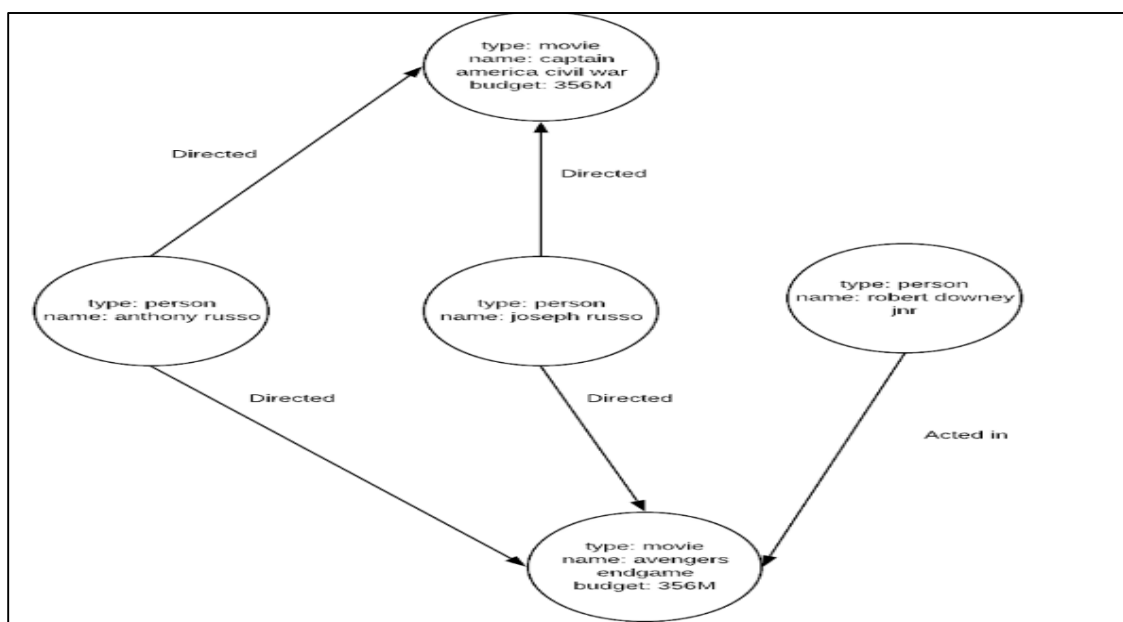


Figure 1: Sample Movie Database

In this graph we can see we're describing a number of entities of differing types. We have 'people', who have typical attributes such as name and gender, and also 'films', which have a title, release date and budget. These are represented in the diagram by bubbles.

Linking the bubbles to give the graph structure are the edges. These describe the relationships between the linked nodes; for example we can see that Avengers:Endgame we directed by Anthony and Joseph Russo, who also directed Captain America

Neo4J

Neo4j is the pre-eminent graph database engine, offering native graph data storage and processing. It is highly scalable, schema free (NoSQL) and supports ACID rules.

Cypher

Neo4j provides a powerful declarative language known as *Cypher*. Cypher is human-friendly query language that uses ASCII-Art to represent visual graph patterns for finding or updating data in Neo4j.

CREATE

The CREATE clause is used to create nodes and relationships.

Create a Node

Syntax to create a single node

```
$ CREATE (n) return (n)
```

The above create statement will create an empty node and return statement will display the empty node as follows.



Create multiple nodes

Syntax to create multiple nodes

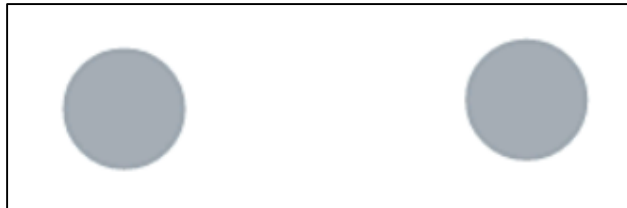
```
$ CREATE (n) , (m)
```

The above create statement will create 2 empty nodes.

You can return the graph as follows.

```
$ Match(n) Return (n)
```

The graph with 2 empty nodes



Create a node with a label

Syntax to create a node with a label

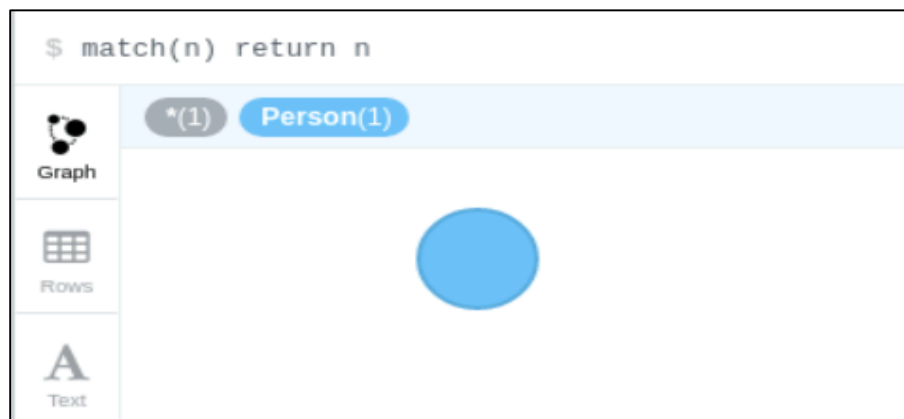
```
$ CREATE (n:Person)
```

The above create statement will create a node with label person.

You can return the graph as follows.

```
$ Match(n) Return (n)
```

The graph with 2 empty nodes



Create node and add labels and properties

Syntax to create a node with a label and properties

```
$ CREATE (node:label {key: value})
```

Example

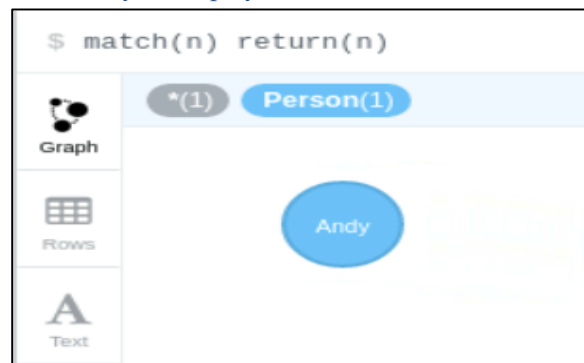
```
$ CREATE (n:Person {name: 'Andy', title: 'Developer'})
```

The above create statement will create a node with label person having properties name as ‘Andy’ and title as ‘Developer’.

You can return the graph as follows.

```
$ Match(n) Return(n)
```

The node with name ‘Andy’ is displayed as follows.



Create Relationship

Syntax to create a relationship between 2 nodes

```
$ CREATE (node1:label1) -[:RelationshipType] -> (node2:label2)
```

This will first create two new nodes and then create a new relationship between them.

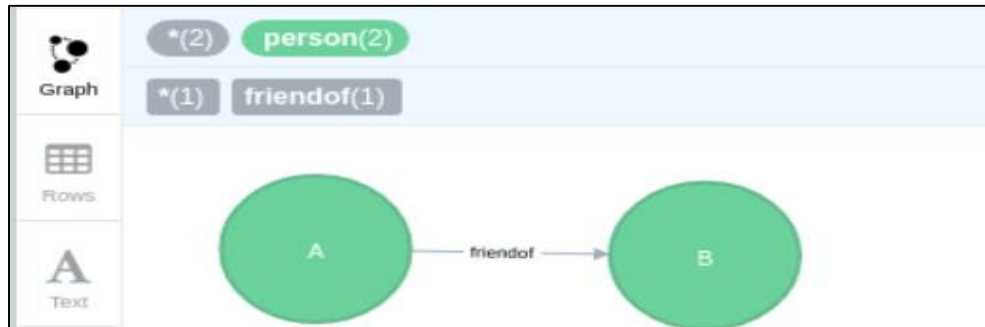
Example

```
$ CREATE (a:person{name:'A'}) -[:friendof] -> (b:person{name:'B'})
```

This will create a *friendof* relationship from person node a to person node b.

You can return the graph as follows.

```
$ Match(n) Return (n)
```



If we need to create a relationship between existing node than first we need to find the nodes using MATCH clause and create a relationship using CREATE clause.

```
$ MATCH(a:label), (b:label) Where a.property = value AND b.property = value  
CREATE (a) - [r:RELTYPE] -> (b)
```

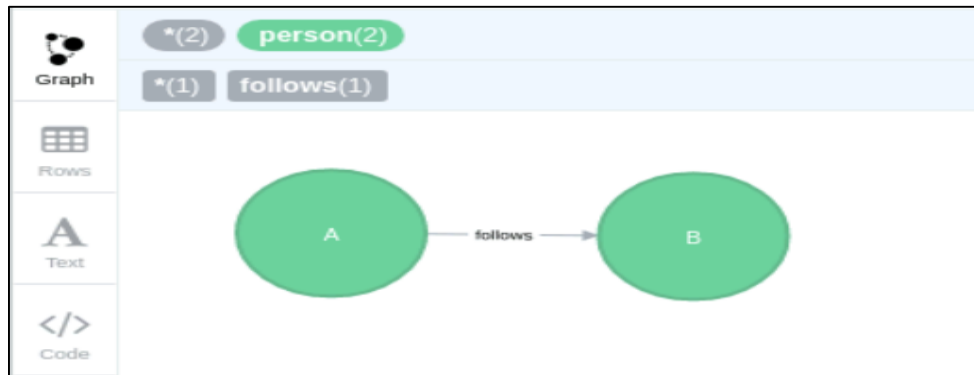
Example

```
$ MATCH(a:person), (b:person) Where a.name = 'A' AND b.name = 'B'  
CREATE (a) - [r:follows] -> (b)
```

The above query will create a follows relationship between 2 person nodes having names A and B

You can return the graph as follows.

```
$ Match(n) Return (n)
```



Create a relationship with given label and properties

syntax

```
$ CREATE (node1)-[label:relationshipType {properties}]->(node)
```

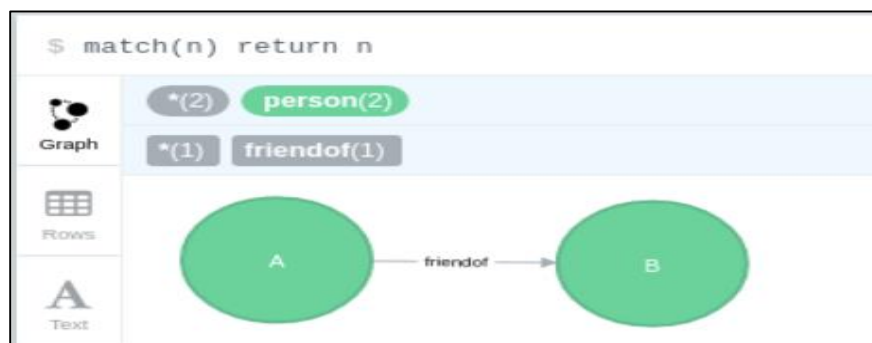
this will create a relationship with given label and properties.

Example

```
$ CREATE (a:person{name:'A'})-[label:friendof{since:2021}]->
(b:person{name:'B'})
```

You can return the graph as follows.

```
$ Match(n) Return (n)
```



MATCH

The **MATCH** clause allows you to specify the patterns Neo4j will search for in the database. This is the primary way of getting data into the current set of bindings.

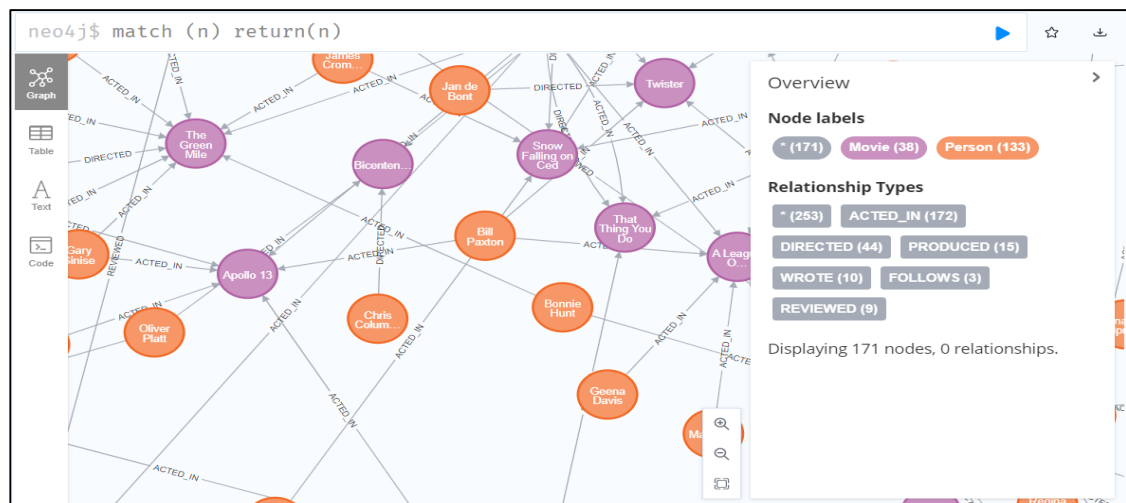
You can access the **Cloud Neo4j sandbox instance** (sandbox.neo4j.com) for movies graph database. One can load the movies database on the instance and run these retrieval queries (match clause) there only.

Get all nodes

By just specifying a pattern with a single node and no labels, all nodes in the graph will be returned.
syntax

```
neo4j$ match (n)
Return (n)
```

The output is as follows



Get all nodes with a label

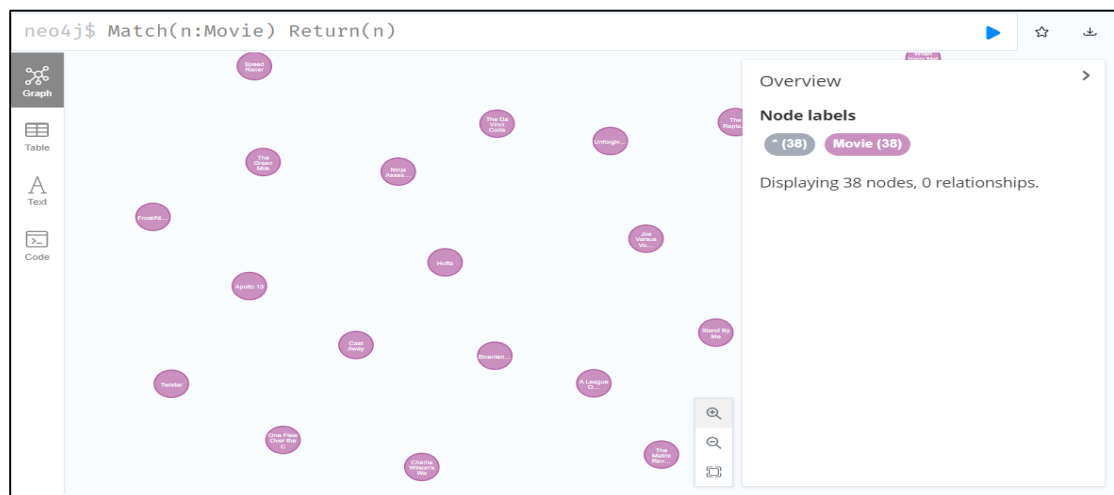
syntax

```
$ Match(n:label)
Return(n)
```

example

```
$ Match(n:Movie)
Return(n)
```

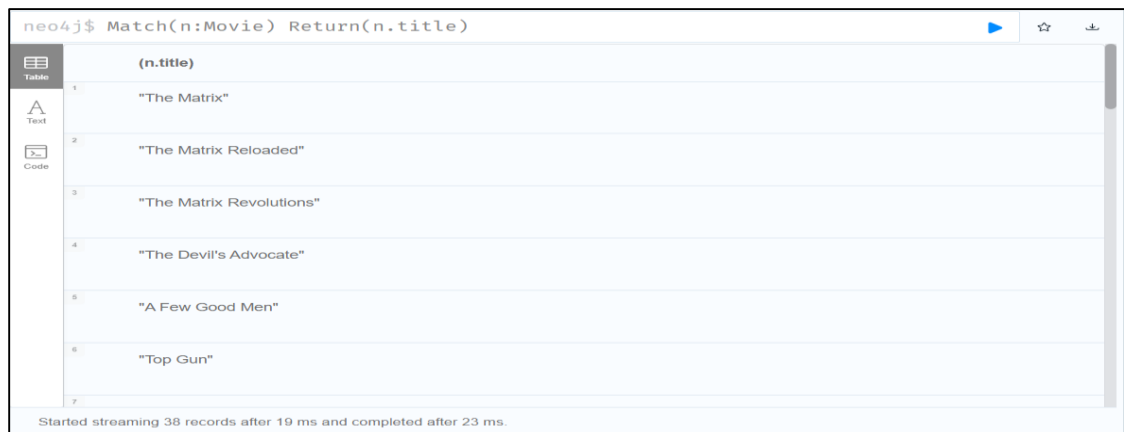
The above query will return all nodes with label movies as follows



In order to return only movie titles, I can specify the same in the return statement as follows

```
$ Match(n:Movie)
Return(n.title)
```

The above query will return all movie titles as follows



The screenshot shows the Neo4j Cypher query interface with the query `neo4j$ Match(n:Movie) Return(n.title)`. The interface displays a table with 38 rows of movie titles. The first six rows are visible in the image. At the bottom, a status message reads: 'Started streaming 38 records after 19 ms and completed after 23 ms.'

	(n.title)
1	"The Matrix"
2	"The Matrix Reloaded"
3	"The Matrix Revolutions"
4	"The Devil's Advocate"
5	"A Few Good Men"
6	"Top Gun"
7	

Get related nodes

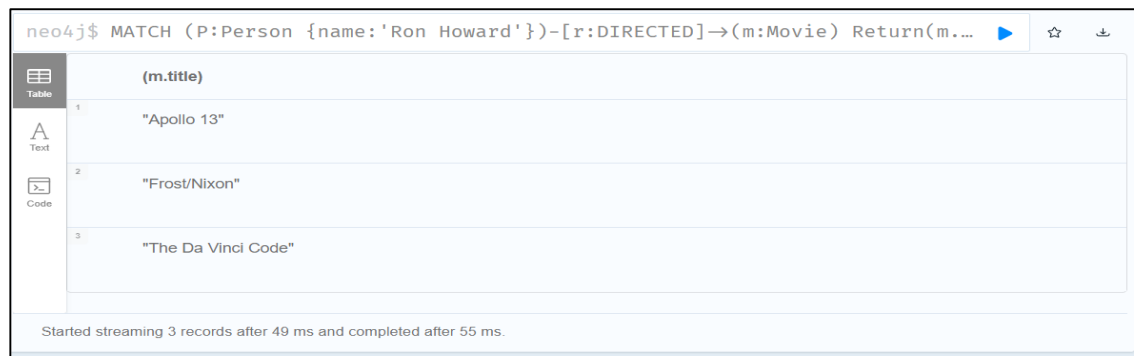
syntax

```
$ MATCH (node:label)<-[:Relationship]-(n)
  Return (n)
```

Example: Return all movies directed by 'Ron Howard'

```
$ MATCH (P:Person {name:'Ron Howard'})-[r:DIRECTED]->(m:Movie)
  Return (m.title)
```

The output of the above query is as below



neo4j\$ MATCH (P:Person {name:'Ron Howard'})-[r:DIRECTED]->(m:Movie) Return(m...

	(m.title)
1	"Apollo 13"
2	"Frost/Nixon"
3	"The Da Vinci Code"

Started streaming 3 records after 49 ms and completed after 55 ms.

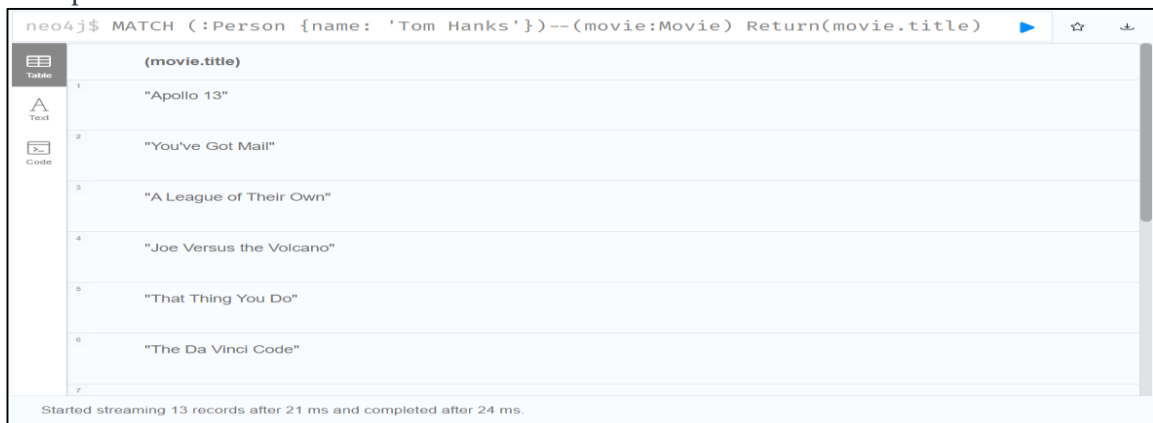
Match with labels

To constraint your pattern with labels on nodes, you add it to your pattern nodes, using the label syntax.

Example: Returns any nodes connected with the Person 'Tom Hanks' that are labeled Movie.

```
$ MATCH (:Person {name: 'Tom Hanks'})--(movie:Movie)
  Return (movie.title)
```

Output



neo4j\$ MATCH (:Person {name: 'Tom Hanks'})--(movie:Movie) Return(movie.title)

	(movie.title)
1	"Apollo 13"
2	"You've Got Mail"
3	"A League of Their Own"
4	"Joe Versus the Volcano"
5	"That Thing You Do"
6	"The Da Vinci Code"
7	

Started streaming 13 records after 21 ms and completed after 24 ms.

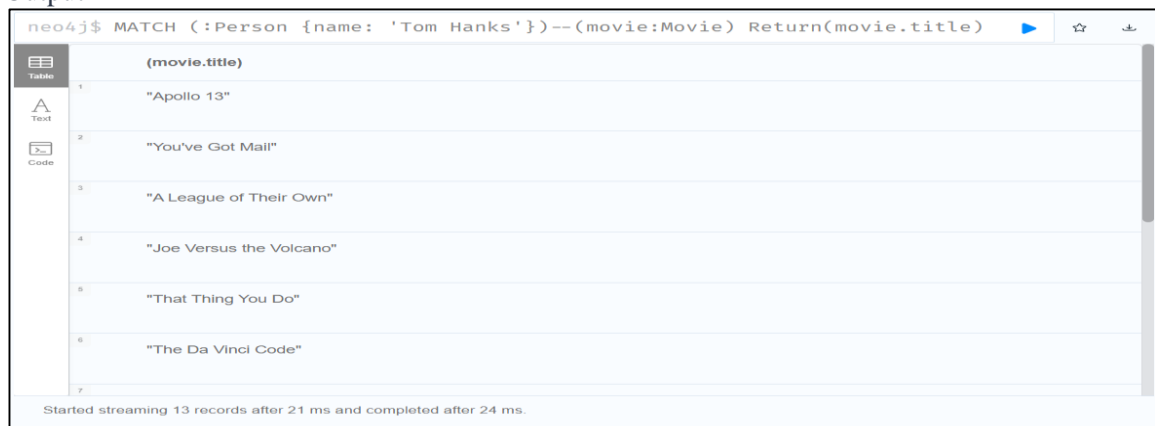
Directional relationships

When the direction of a relationship is of interest, it is shown by using `-->` or `<--`, like this:

Example: Returns any nodes connected with the Person 'Tom Hanks' by an outgoing relationship

```
$ MATCH (:Person {name: 'Tom Hanks'})-->(movie:Movie)
Return(movie.title)
```

Output



neo4j\$ MATCH (:Person {name: 'Tom Hanks'})--(movie:Movie) Return(movie.title)

	(movie.title)
1	"Apollo 13"
2	"You've Got Mail"
3	"A League of Their Own"
4	"Joe Versus the Volcano"
5	"That Thing You Do"
6	"The Da Vinci Code"
7	

Started streaming 13 records after 21 ms and completed after 24 ms.

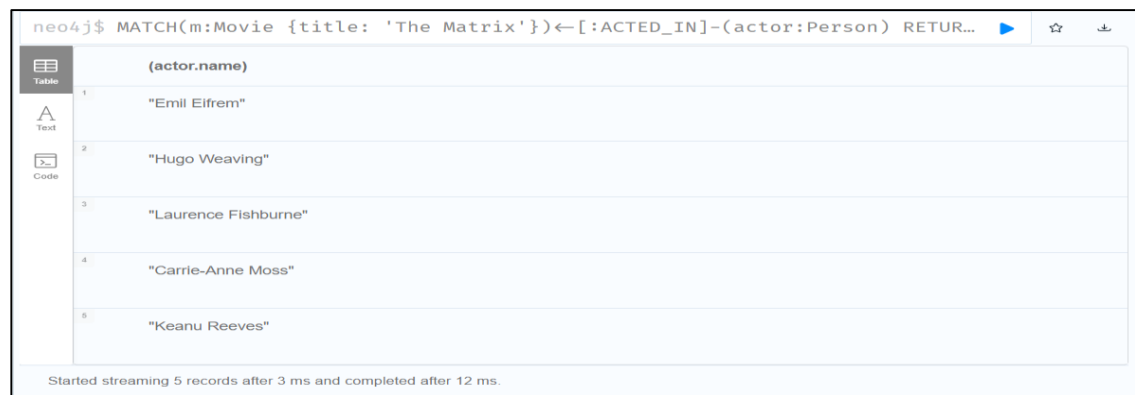
Match on relationship type

When you know the relationship type you want to match on, you can specify it by using a colon together with the relationship type.

Example: Returns all actor names who acted in movie 'Apollo13'

```
$ MATCH(m:Movie {title: 'The Matrix'})<-[ACTED_IN]-(actor:Person)
RETURN(actor.name)
```

The output is as follows



neo4j\$ MATCH(m:Movie {title: 'The Matrix'})<-[ACTED_IN]-(actor:Person) RETURN...

	(actor.name)
1	"Emil Eifrem"
2	"Hugo Weaving"
3	"Laurence Fishburne"
4	"Carrie-Anne Moss"
5	"Keanu Reeves"

Started streaming 5 records after 3 ms and completed after 12 ms.

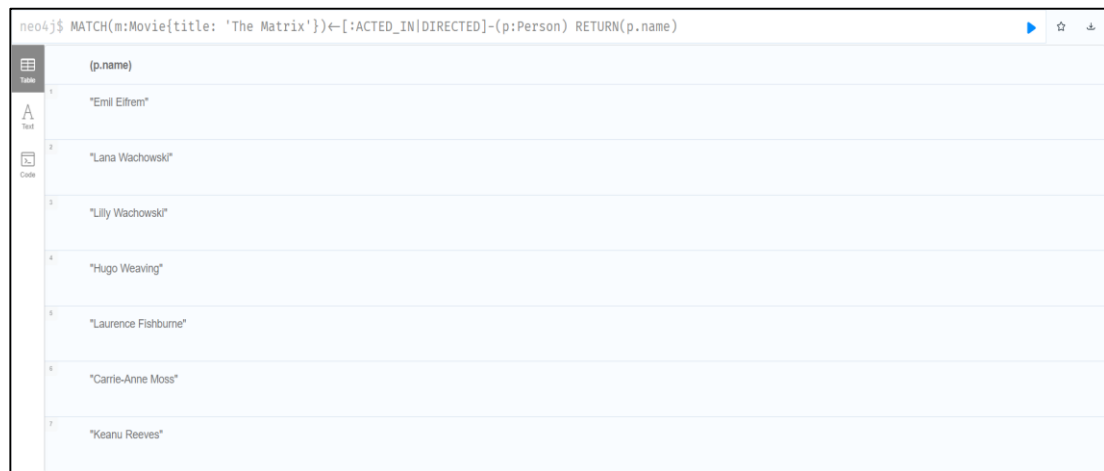
Match on multiple relationship types

To match on one of multiple types, you can specify this by chaining them together with the pipe symbol (|)

Example: Returns nodes with an **ACTED_IN** or **DIRECTED** relationship to 'The Matrix'.

```
$ MATCH (m:Movie{title: 'The Matrix'})<-[:ACTED_IN|DIRECTED]-(p:Person)
RETURN (p.name)
```

The output of the query is as follows



neo4j\$ MATCH (m:Movie{title: 'The Matrix'})<-[:ACTED_IN|DIRECTED]-(p:Person) RETURN (p.name)

	(p.name)
1	"Emil Eifrem"
2	"Lana Wachowski"
3	"Lilly Wachowski"
4	"Hugo Weaving"
5	"Laurence Fishburne"
6	"Carrie-Anne Moss"
7	"Keanu Reeves"

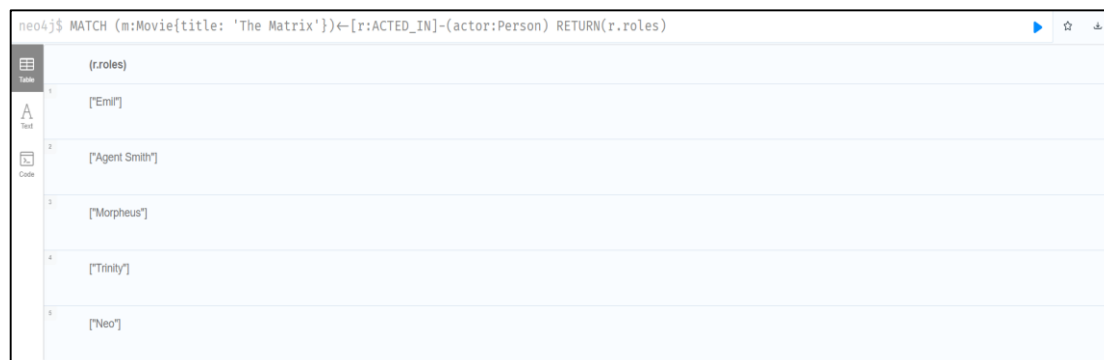
Match on relationship type and use a variable

If you both want to introduce a variable to hold the relationship, and specify the relationship type you want, just add them both, like this:

Example: Returns **ACTED_IN** roles for 'The Matrix'.

```
$ MATCH (m:Movie{title: 'The Matrix'})<-[r:ACTED_IN]-(actor:Person)
RETURN (r.roles)
```

Output



neo4j\$ MATCH (m:Movie{title: 'The Matrix'})<-[r:ACTED_IN]-(actor:Person) RETURN (r.roles)

	(r.roles)
1	["Emil"]
2	["Agent Smith"]
3	["Morpheus"]
4	["Trinity"]
5	["Neo"]

Multiple relationships

Relationships can be expressed by using multiple statements in the form of ()--()

Example: Returns the movies **'Tom Hanks'** acted in and its director.

```
$ MATCH (p:Person {name: 'Tom Hanks'})-[:ACTED_IN]->(m:Movie)<-[:DIRECTED]-(director:Person)
RETURN m.title, director.name
```

The output is as follows

neo4j\$ MATCH (p:Person {name: 'Tom Hanks'})-[:ACTED_IN]->(m:Movie)<-[:DIRECTED]-(director:Person) RETURN m.title, director.name

	m.title	director.name
1	"Apollo 13"	"Ron Howard"
2	"You've Got Mail"	"Nora Ephron"
3	"A League of Their Own"	"Penny Marshall"
4	"Joe Versus the Volcano"	"John Patrick Stanley"
5	"That Thing You Do"	"Tom Hanks"
6	"The Da Vinci Code"	"Ron Howard"
7	"Cloud Atlas"	"Tom Tykwer"
8	"Cloud Atlas"	"Lana Wachowski"
9	"Cloud Atlas"	"Lilly Wachowski"

Started streaming 14 records after 49 ms and completed after 55 ms.

REMOVE

The **REMOVE** clause is used to remove properties and labels from nodes and relationships.

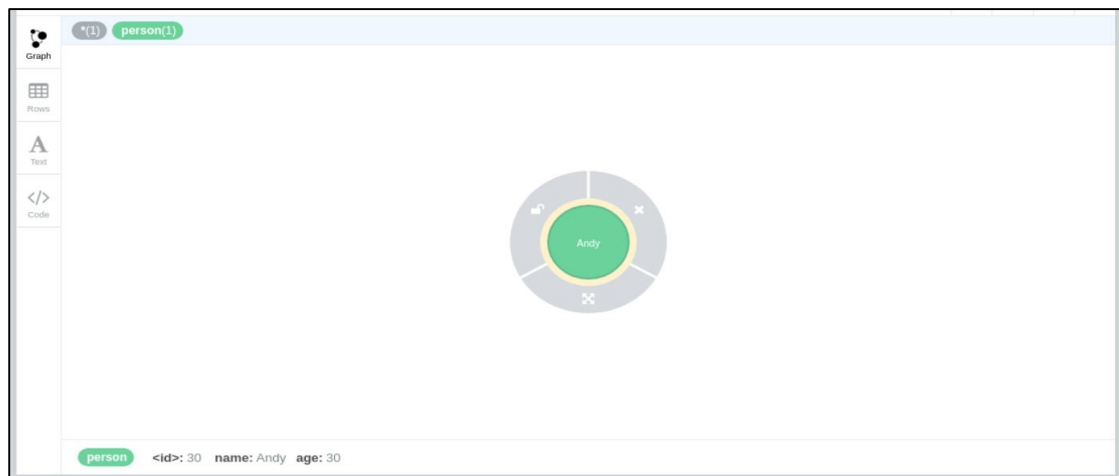
Remove a property

Example: remove age property of a node

```
$ MATCH (a:{name:'Andy'})
  REMOVE a.age
  RETURN a.name,a.age
```

The node is returned, and no property age exists on it.

Existing node with label person having name as Andy and age 30 as properties



You can remove the age by executing above query and list the graph.

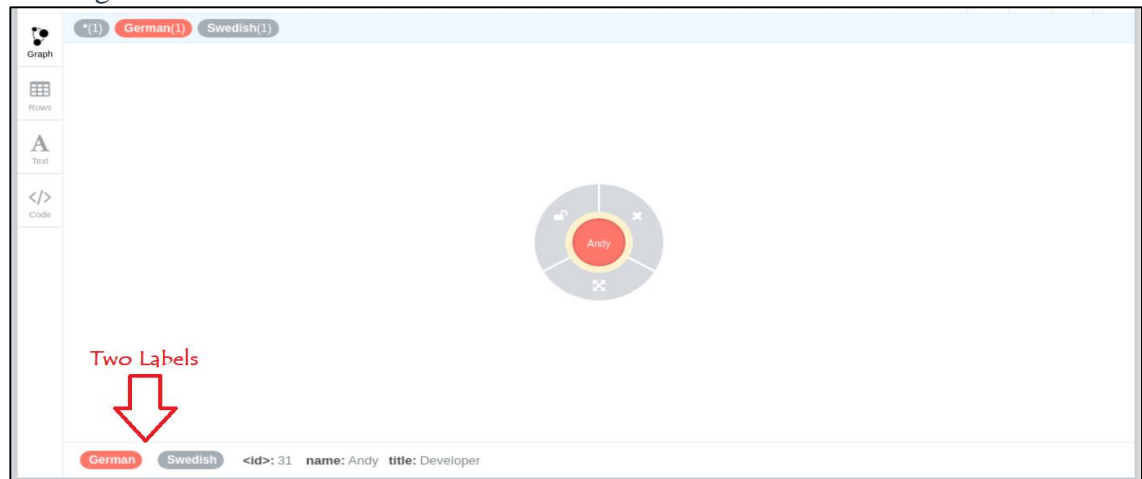


Remove a Label

Example: remove label *German* from node name as 'Andy'

```
$ MATCH (n:German{name:'Andy'})
  REMOVE n:German
  RETURN n.name, labels(n)
```

Existing node with 2 labels German and Swedish



You can remove the label by executing above query and list the graph.



Remove multiple labels from a node

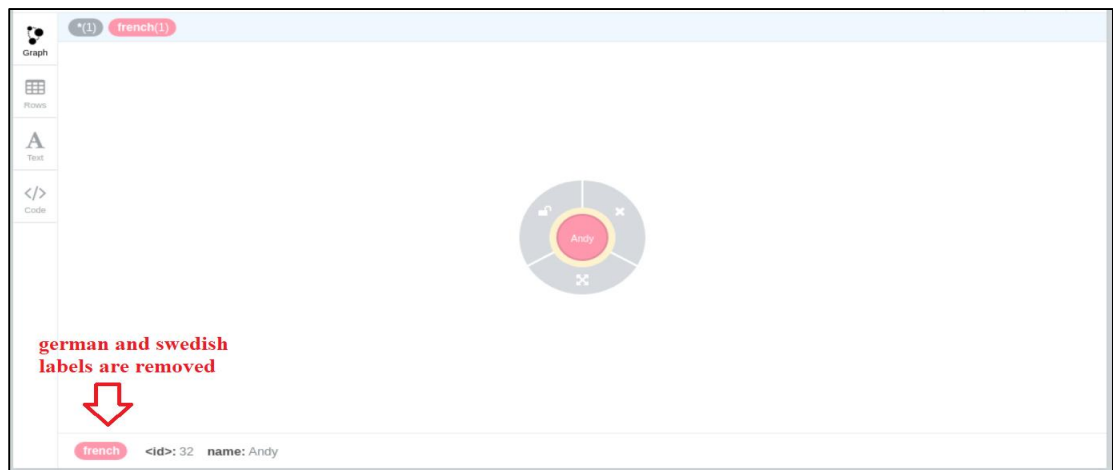
Example: remove label *German* and *Swedish* from node name as peter

```
$ MATCH (n:french:{name:'Andy'})
REMOVE n:German:Swedish
RETURN n.name, labels(n)
```

Existing node with 3 labels French, German and Swedish



You can remove multiple labels by executing above query and list the graph.



DELETE

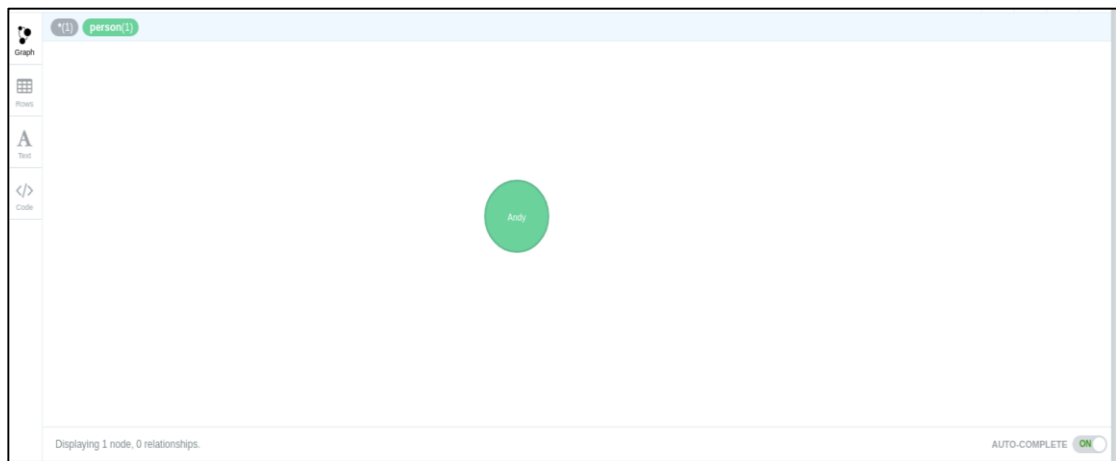
The DELETE clause is used to delete nodes, relationships and paths

Delete a node

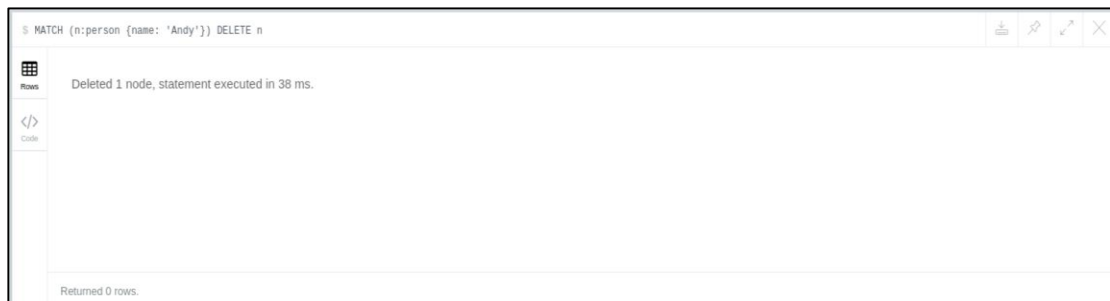
Example: Delete a node with name Andy

```
$ MATCH (n:Person {name: 'Andy'})  
DELETE n
```

Existing node with name 'Andy'



You can delete the node by executing the above delete query.



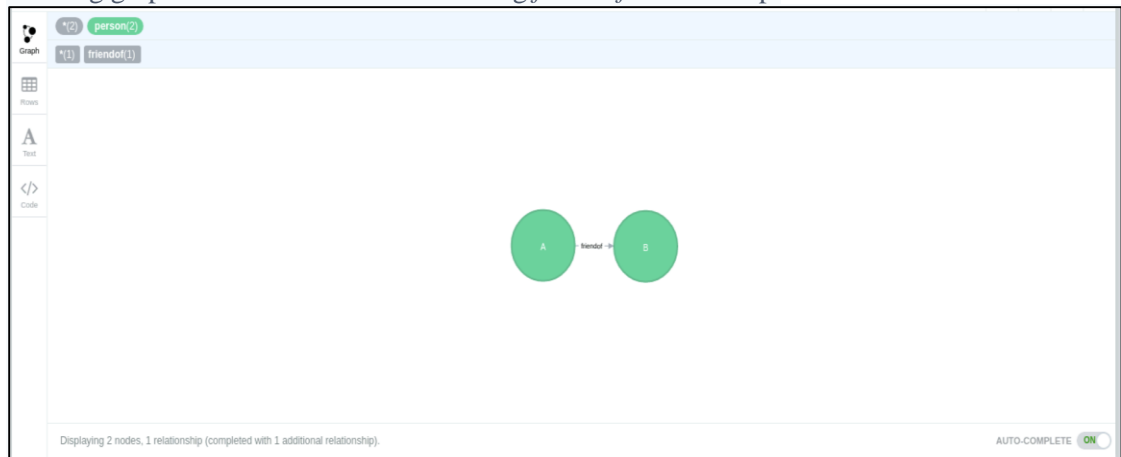
Delete all nodes and relationships

Example

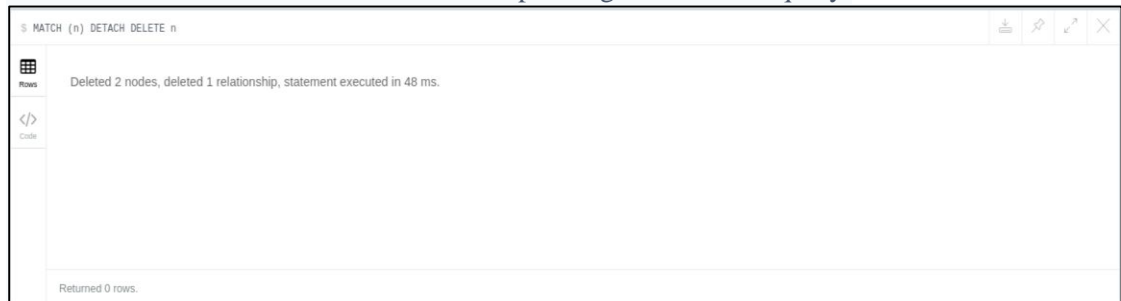
```
$ MATCH (n)
  DETACH DELETE n
```

you cannot delete a node without also deleting relationships that start or end on said node.
Either explicitly delete the relationships, or use DETACH DELETE

Existing graph with 2 nodes A and B having *friendof* relationship.



You can delete the all nodes and relationships using above delete query.

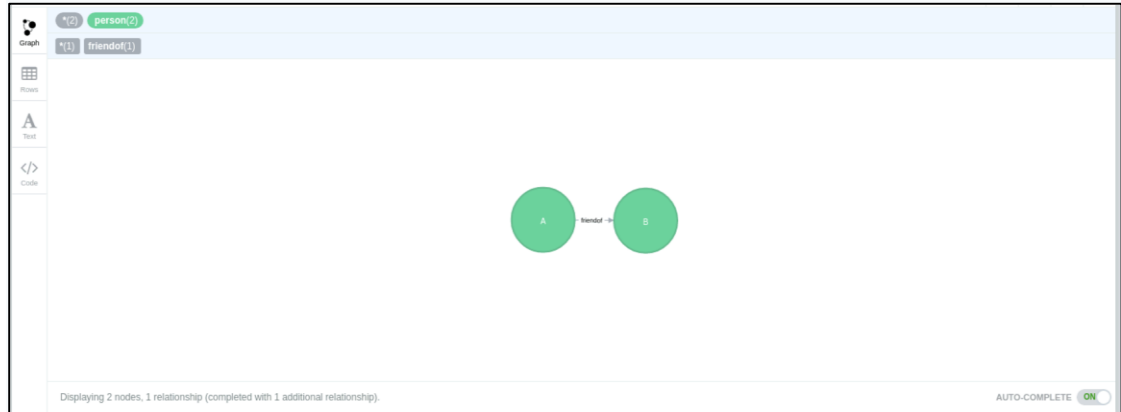


Delete a node with all its relationships

Example

```
$ MATCH (n:person{name: 'A'})
DETACH DELETE n
```

Existing graph with 2 nodes A and B having *friendof* relationship.

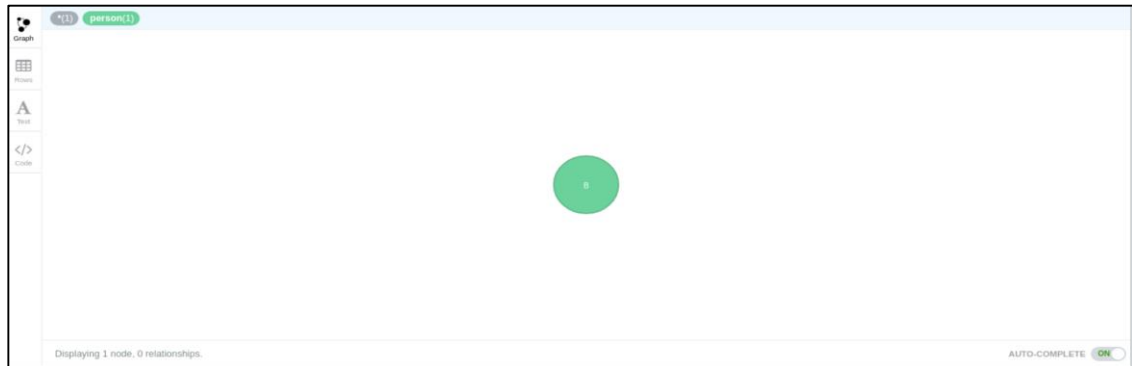


You can delete node A and relationship using above delete query.



You can display the graph as follows

```
$ MATCH (n) RETURN n
```

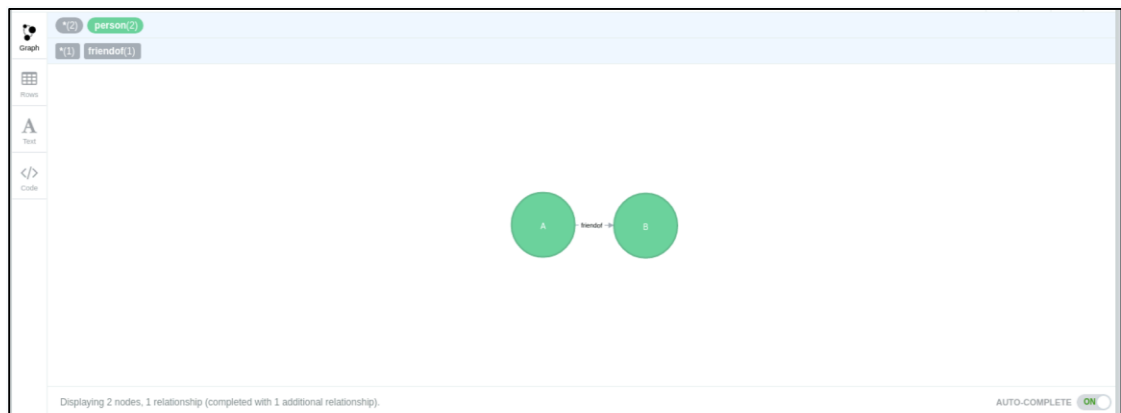


Delete relationships only

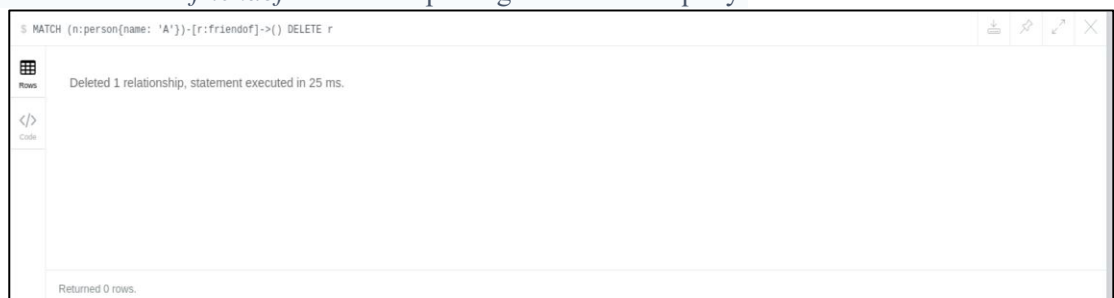
Example

```
$ MATCH (n:person{name: 'A'})-[r:friendof]->()
DELETE r
```

Existing graph with 2 nodes A and B having *friendof* relationship.

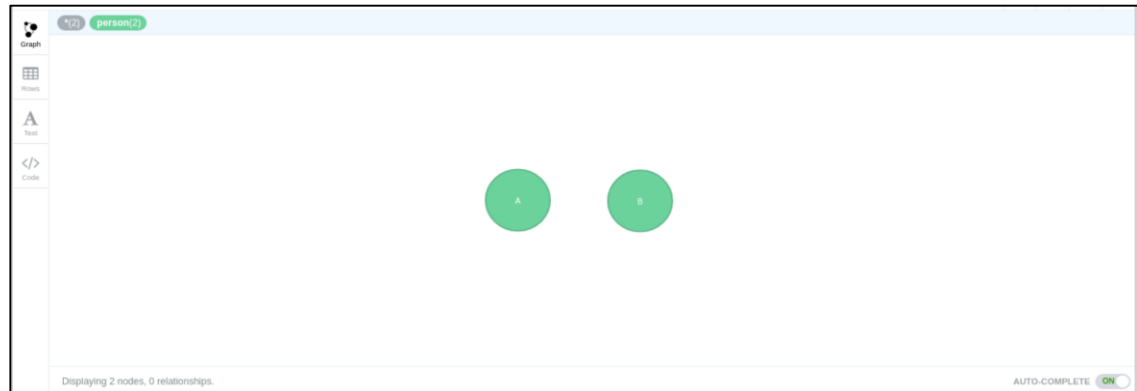


You can delete '*friendof*' relationship using above delete query.



You can display the graph as follows

```
$ MATCH (n) RETURN n
```



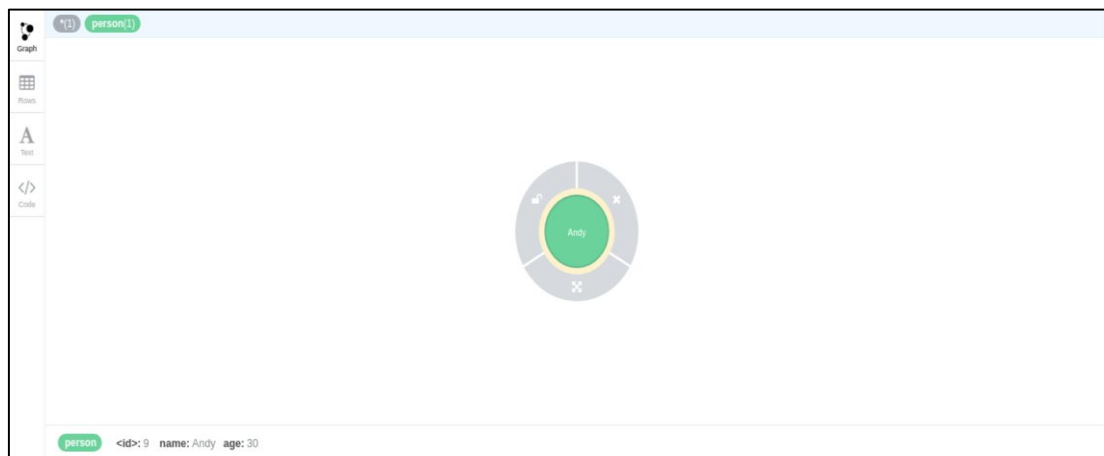
SET

SET can be used to update a property on a node or relationship.

Example: updating a property type

```
$ MATCH (n:person {name: 'Andy'})
  SET n.age = toString(n.age)
  RETURN n.name, n.age
```

Existing node with two properties name: 'Andy' and age:30 where age is integer



The above set query will update the type of age to string

<pre>\$ MATCH (n:person {name: 'Andy'}) SET n.age = toString(n.age) RETURN n.name, n.age</pre>	
n.name	n.age
Andy	30
Set 1 property, returned 1 row in 24 ms.	

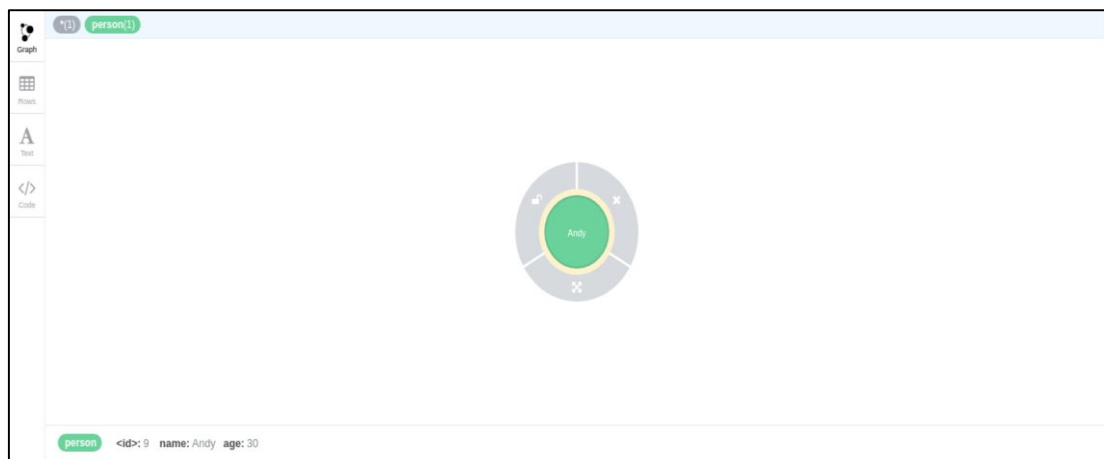
Add a property using SET

Example

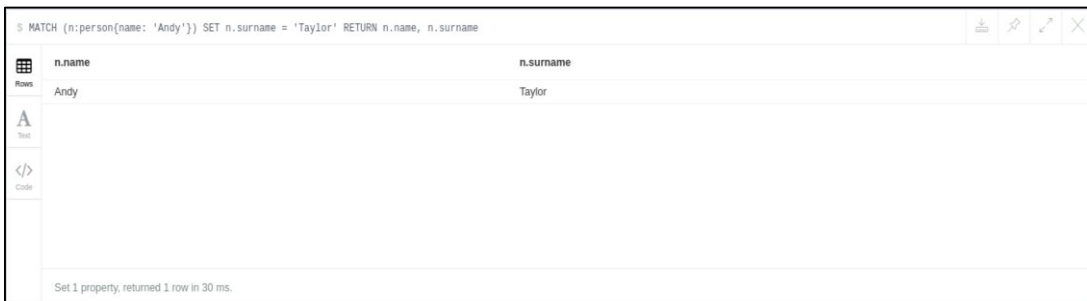
```
$ MATCH (n:person{name: 'Andy'})
  SET n.surname = 'Taylor'
  RETURN n.name, n.surname
```

In case surname property does not exist on node with name Andy it will be added

Existing node with two properties name: 'Andy' and age:30



The above query will add surname property with its value 'Taylor'

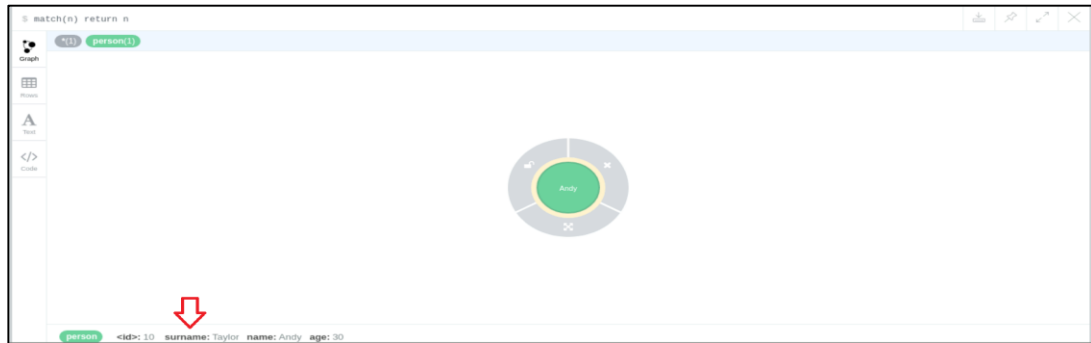


The screenshot shows the Cypher Studio interface with the query: `MATCH (n:person{name: 'Andy'}) SET n.surname = 'Taylor' RETURN n.name, n.surname`. The result is displayed in a table with two columns: `n.name` and `n.surname`. The row shows 'Andy' and 'Taylor'.

n.name	n.surname
Andy	Taylor

Set 1 property, returned 1 row in 30 ms.

You can display the graph and check the property added to the node



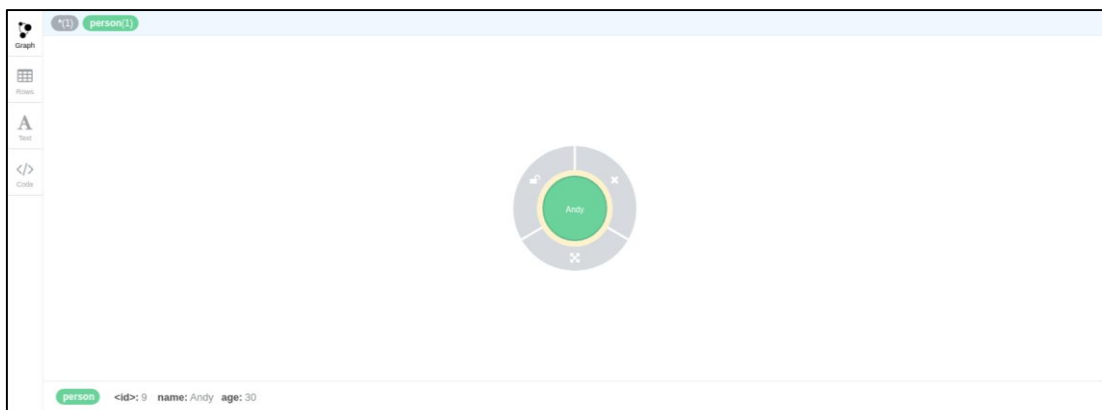
Remove a property using SET

Example

```
$ MATCH (n {name: 'Andy'})
  SET n.name = null
  RETURN n.name, n.age
```

you cannot delete a node without also deleting relationships that start or end on said node.
Either explicitly delete the relationships, or use DETACH DELETE

Existing node with two properties name: 'Andy' and age:30

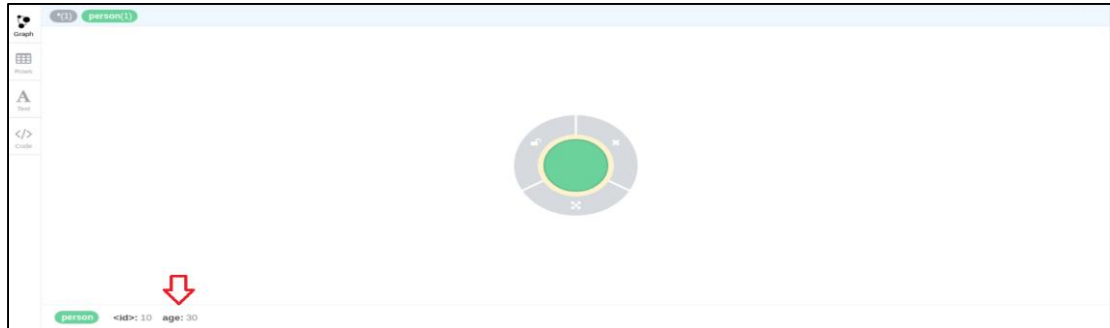


The above query will remove property name from the node

n.name		n.age
null		30

Set 1 property, returned 1 row in 37 ms.

You can display the graph and check the properties



Outputs/Results

- Students should be able to appreciate the usage of Neo4J graph database.
- Students should be able to perform CRUD operations using cypher queries

Observations

Students should carefully observe the syntax of cypher queries and observe the output of retrieval queries



REFERENCES

- [Introduction to Neo4J](#)
- [Cypher Manual](#)
- [Introduction to graph databases](#)