COMP1835

Lab 8 Neo4J

Overview:

In this lab you will be working with Neo4J data store using Neo4J Browser and Cypher language. You will model a social network and will create a graph.

You will be using your second VM- COMP1835-graphdb

Login: **username**=student, **password**=student

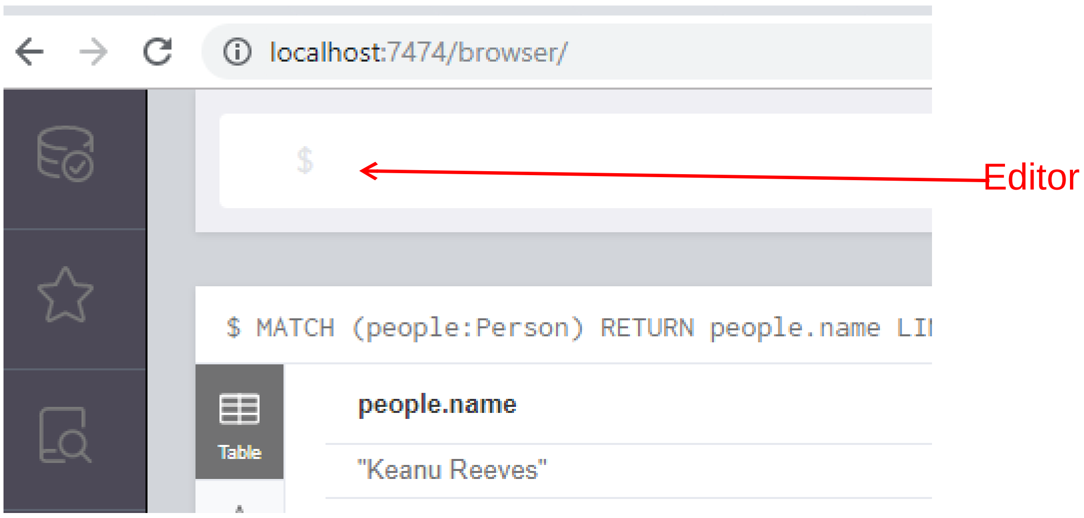
Lab 8.1 Get familiar with Neo4J Browser.

Neo4j Browser is a command driven client, like a web-based shell environment. It is perfect for running ad-hoc graph queries, with just enough ability to prototype a Neo4j-based application.

To open, click on the icon on your desktop.

Editor

The editor is the primary interface for entering and running commands. Enter Cypher queries to work with graph data. Use client-side commands like:help for other operations.

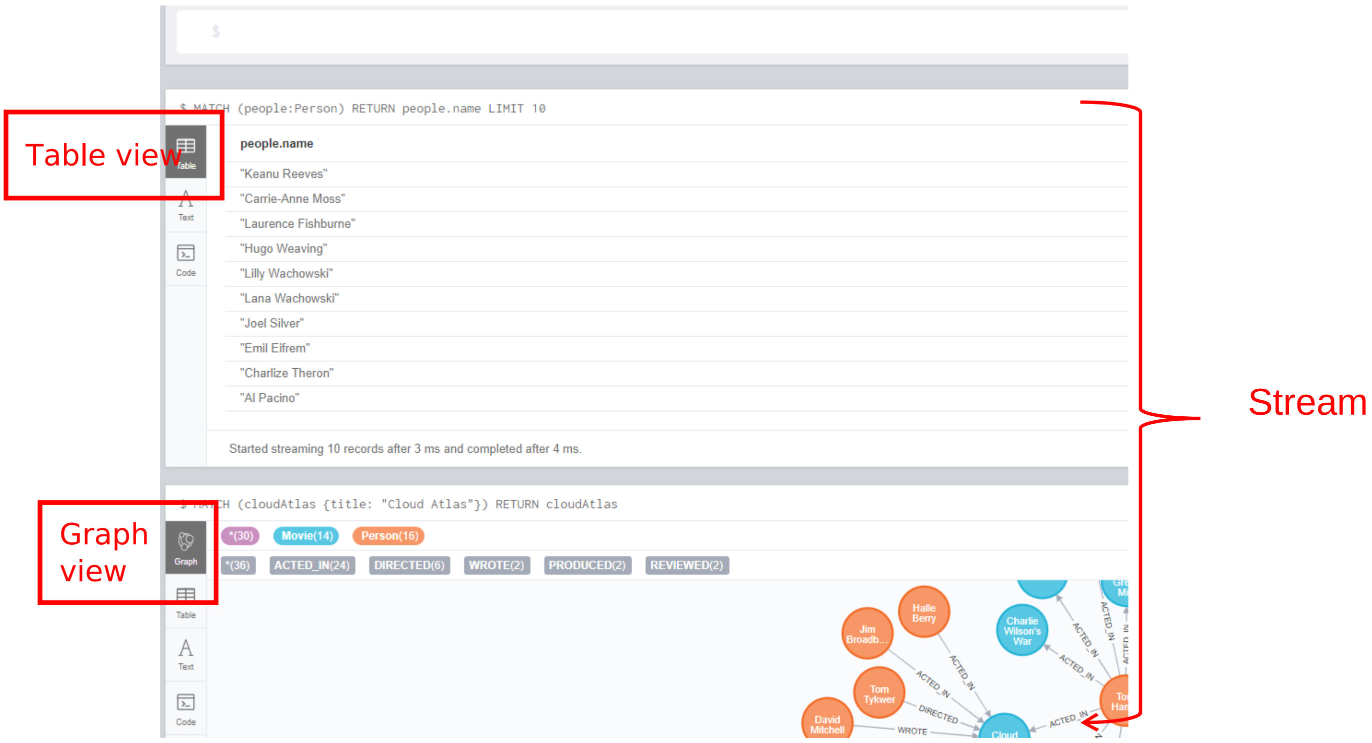


```
$ MATCH (people:Person) RETURN people.name LI
```

people.name
"Keanu Reeves"

Stream

Is scrolling series of result frames, you can clear the stream with the :clear command


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

people.name
"Keanu Reeves"
"Carrie-Anne Moss"
"Laurence Fishburne"
"Hugo Weaving"
"Lilly Wachowski"
"Lana Wachowski"
"Joel Silver"
"Emil Eifrem"
"Charlize Theron"
"Al Pacino"

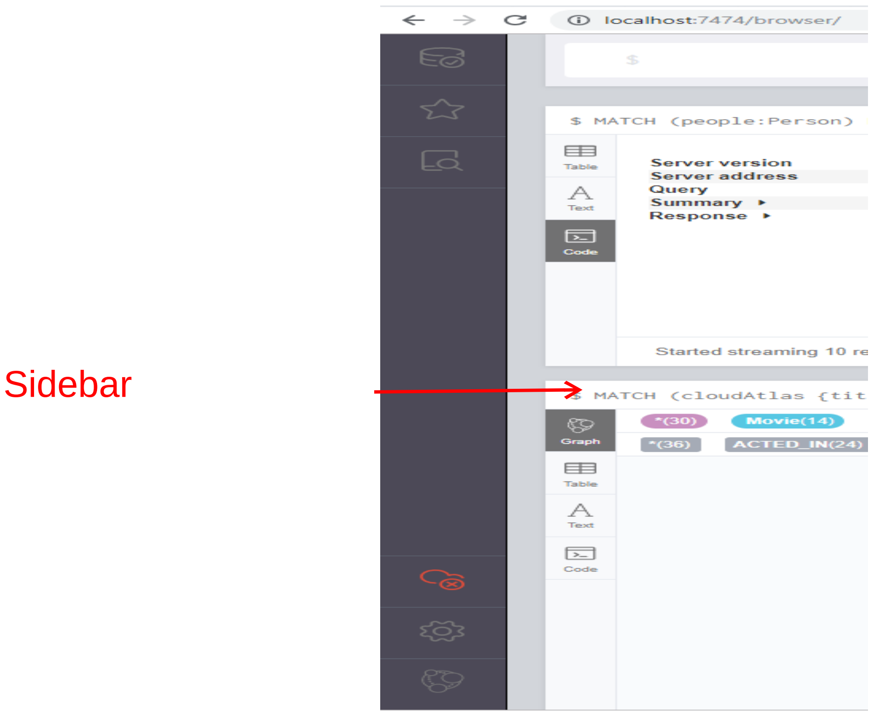
Started streaming 10 records after 3 ms and completed after 4 ms.

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

Sidebar

The sidebar expands to reveal different functional panels for common queries and information.

- Database metadata and basic information
- Saved scripts organized into folders
- Information links for docs and reference
- Credits and licensing information


localhost:7474/browser/

```
$ MATCH (people:Person)
```

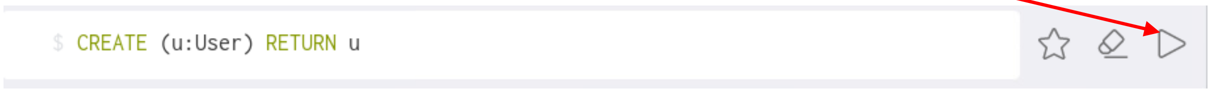
Server version
Server address
Query
Summary ▸
Response ▸

Started streaming 10 re

```
$ MATCH (cloudAtlas {tit
```

Lab 8.2 Creating nodes

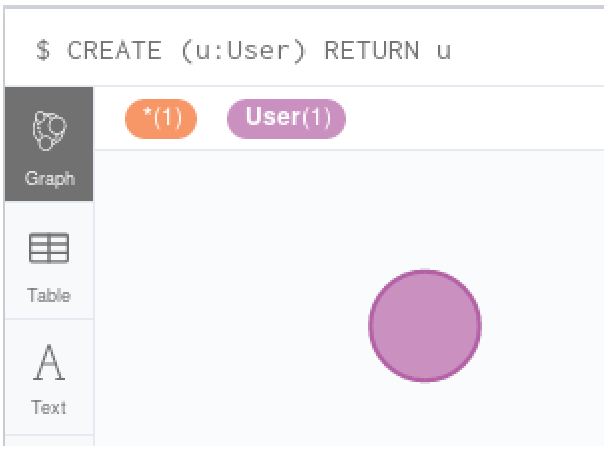
As we modeling a social network, a user registered in the social network is a node. Let's create a new one using the following command in the Editor and press Play button



```
$ CREATE (u:User) RETURN u
```

By adding the RETURN clause, we get a result that consists of one row: the node that was created earlier.

The result is shown in the graph stream:

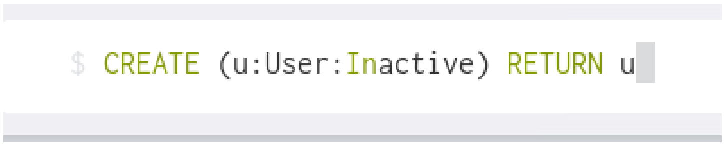


```
$ CREATE (u:User) RETURN u
```

Multiple labels
You can add as many labels as you need to a node by chaining them in a single definition. The following query creates another user with two labels, **User** and **Inactive**

```
CREATE (u:User:Inactive)
RETURN u
```

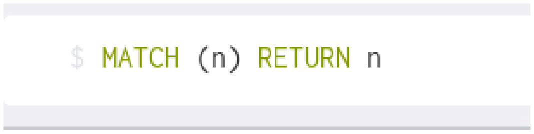
The result is a new node.



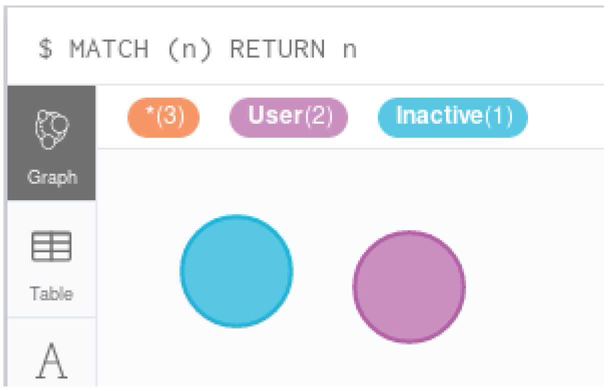
```
$ CREATE (u:User:Inactive) RETURN u
```

To access all nodes in the graph, kind of like a SELECT * FROM entire graph you use statement

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



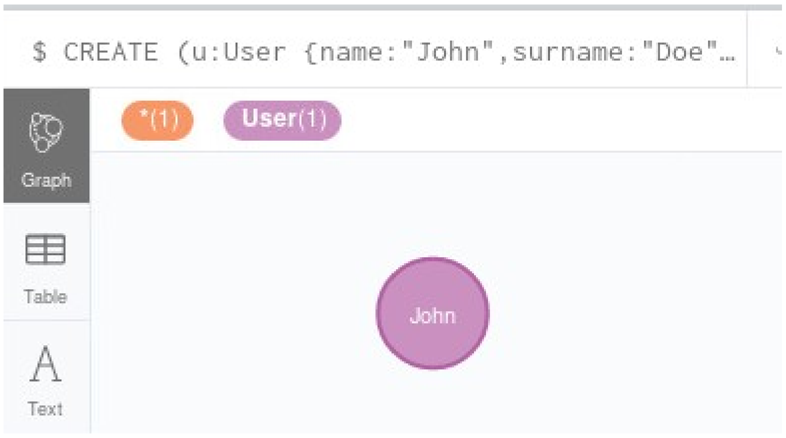
```
$ MATCH (n) RETURN n
```


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

Properties

The nodes we have created so far have no properties. The **`CREATE`** clause supports the creation of properties along with their nodes in a unique query. Create a new node with two properties of name and surname:

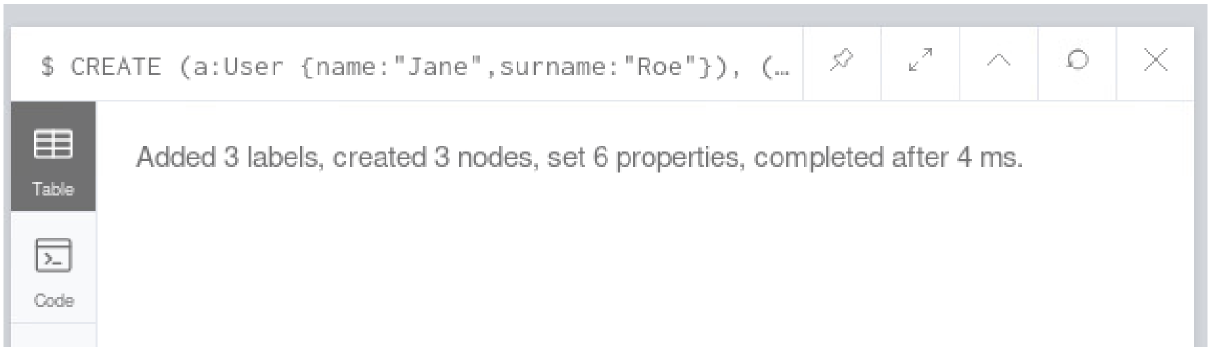
```
CREATE (u:User {name: "John", surname: "Doe"})
RETURN u
```


```
$ CREATE (u:User {name:"John",surname:"Doe"…
```

Lab 8.3 Creating multiple nodes

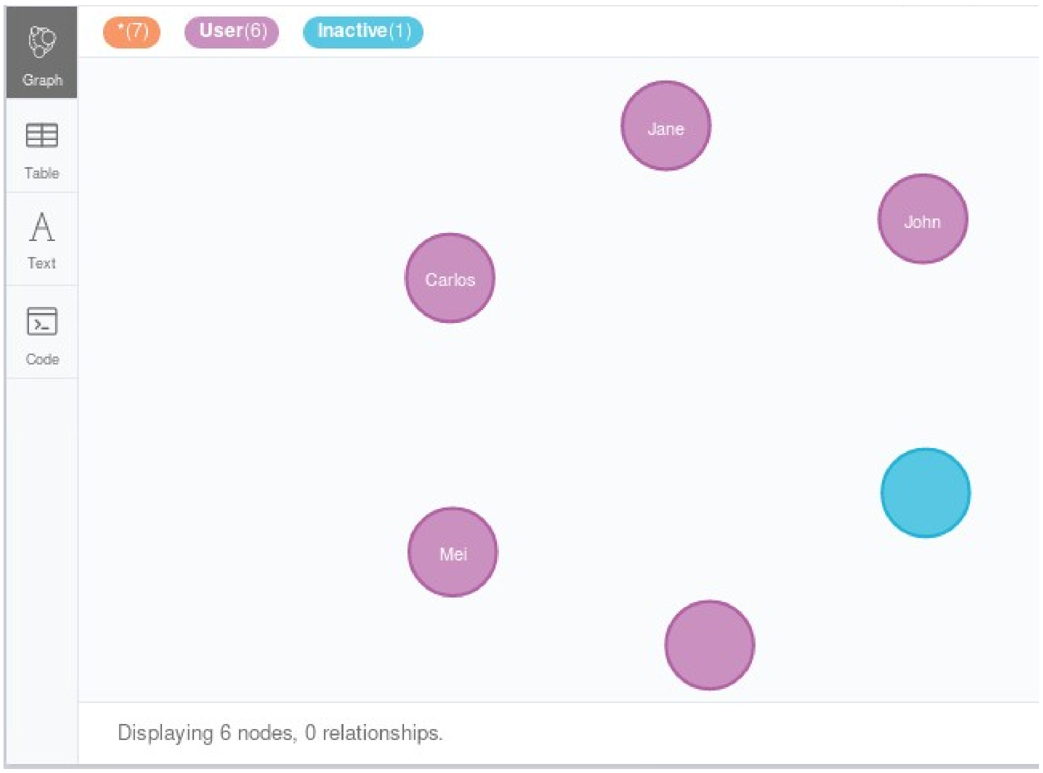
By separating patterns with a comma, you get them to be treated separately resulting into the creation of multiple patterns. Let's create creates three users in a single call:

1. name=Jane, surname=Roe
2. name Carlos, surname=Garcia
3. name=Mei, surname =Weng


```
$ CREATE (a:User {name:"Jane",surname:"Roe"}), (…
```
Added 3 labels, created 3 nodes, set 6 properties, completed after 4 ms.

No result is returned, since the **RETURN** clause is missing.

Use appropriate command to see all nodes:

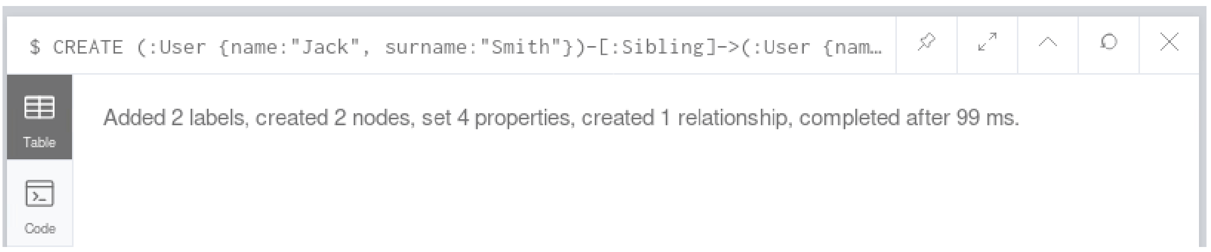


Displaying 6 nodes, 0 relationships.

Lab 8.4 Creatatng relashionships along with nodes

If you want to create relationships along with nodes, just use the relationship pattern, as shown in the following query:

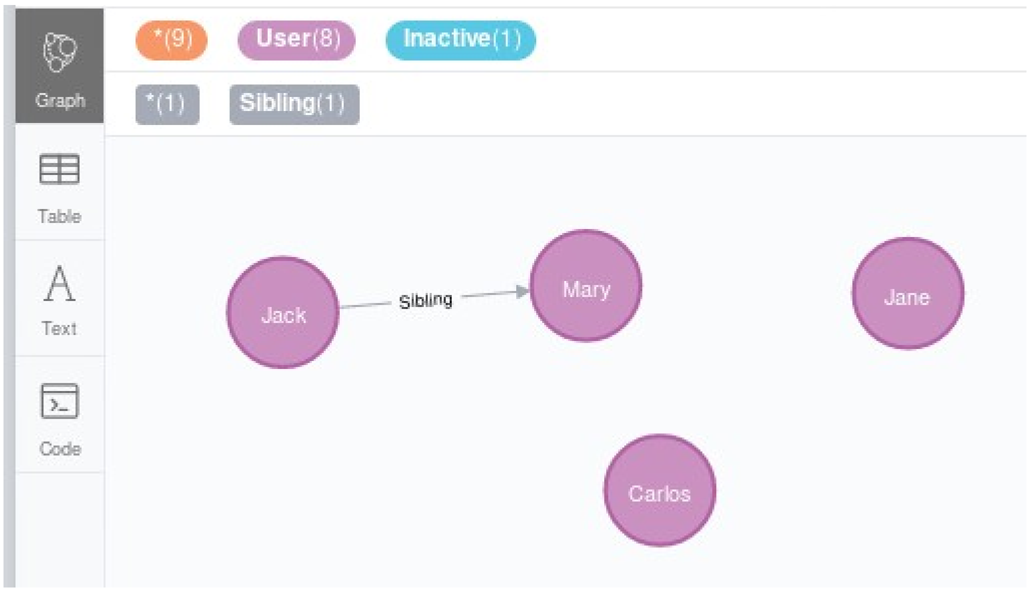
```
CREATE (:User {name: "Jack", surname: "Smith"})
-[:Sibling]->
(:User {name: "Mary", surname: "Smith"})
```



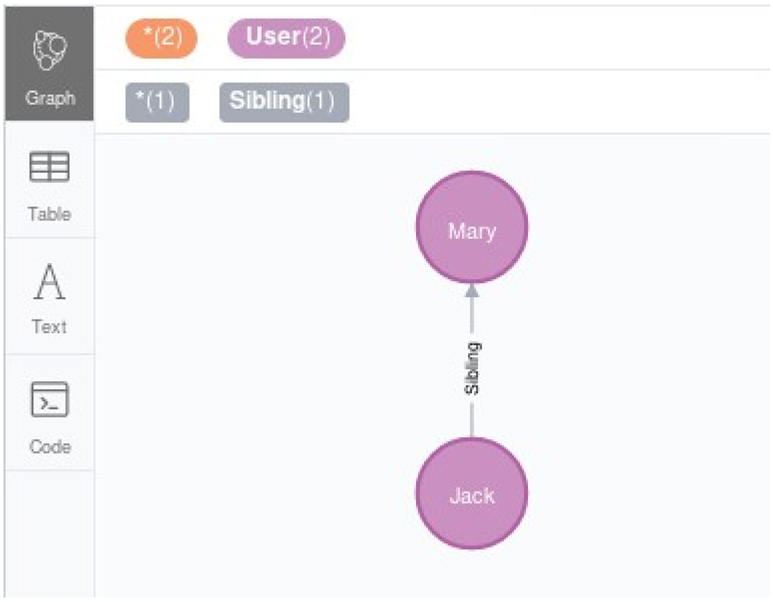
```
$ CREATE (:User {name:"Jack", surname:"Smith"})-[:Sibling]->(:User {nam…
```

Added 2 labels, created 2 nodes, set 4 properties, created 1 relationship, completed after 99 ms.

Use appropriate command to see all nodes:



Return nodes Mary and Jack and relationship between them:



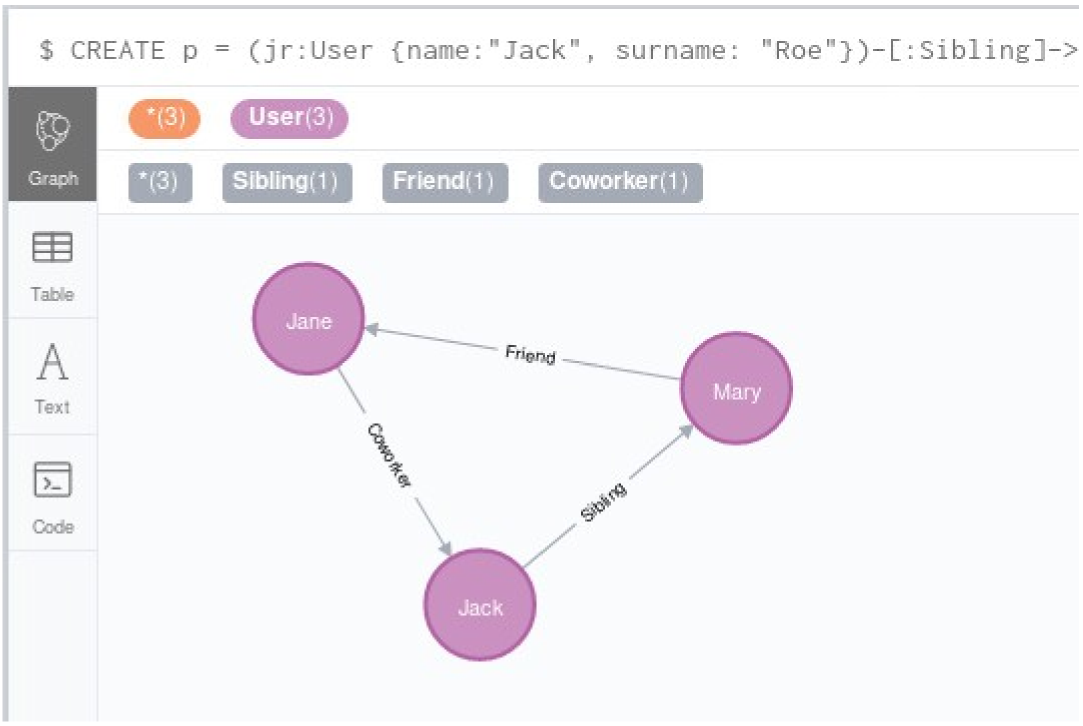
Lab 8.5 Creating full paths

Using a path pattern, you can create a full path in Neo4j. The following CREATE command creates three nodes and then three relationships among them: **`Sibling`**, **`Friend`**, and **`Coworker`**.:

```
$ CREATE p = (jr:User {name:"Jack", surname: "Roe"})-[:Sibling]->
  (:User {name:"Mary", surname:"Roe"})-[:Friend]-> (:User
  {name:"Jane",surname:"Jones"})-[:Coworker {company: "Acme Inc."}]
  -> (jr) RETURN p
```

Relationship **Coworker** has the company name “**Acme Inc"** as its property, and the end node is the first node of the path (the user Jack Roe).

As the query returns the path, Neo4j Browser shows a graph of the whole path:



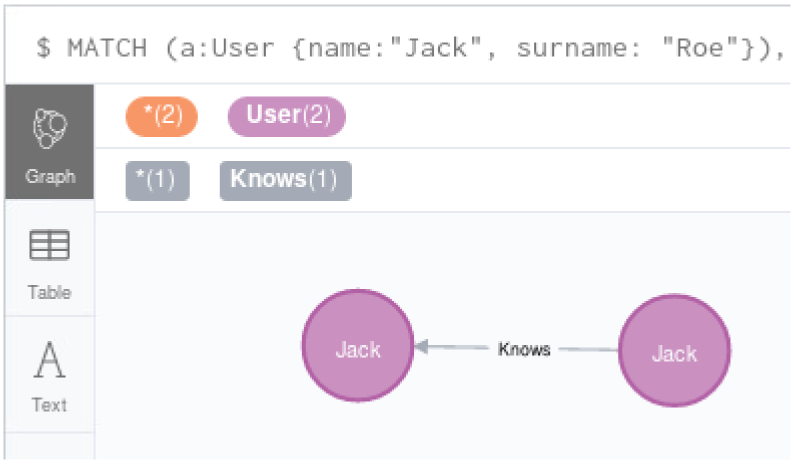
```
$ CREATE p = (jr:User {name:"Jack", surname: "Roe"})-[:Sibling]->
```

Lab 8.6 Creating relationships between existing nodes using read-and-write queries

You can use the **CREATE** clause in conjunction with the **MATCH** statement to add relationships between existing nodes. For example, the following query creates a relationship between two nodes:

```
⚠ $ MATCH (a:User {name:"Jack", surname: "Roe"}), (b:User {name:"Jack",
    surname:"Smith"}) CREATE (a)-[r:Knows]->(b) RETURN a,r,b
```

The **MATCH** statement matches the user nodes Jack Roe and Jack Smith with the variables a and b; then, the **CREATE** clause creates a relation of the type **Knows** between them. Finally, both the user nodes and their new relationship are returned as output.



```
$ MATCH (a:User {name:"Jack", surname: "Roe"}),
```

Lab 8.7 Setting properties and labels.

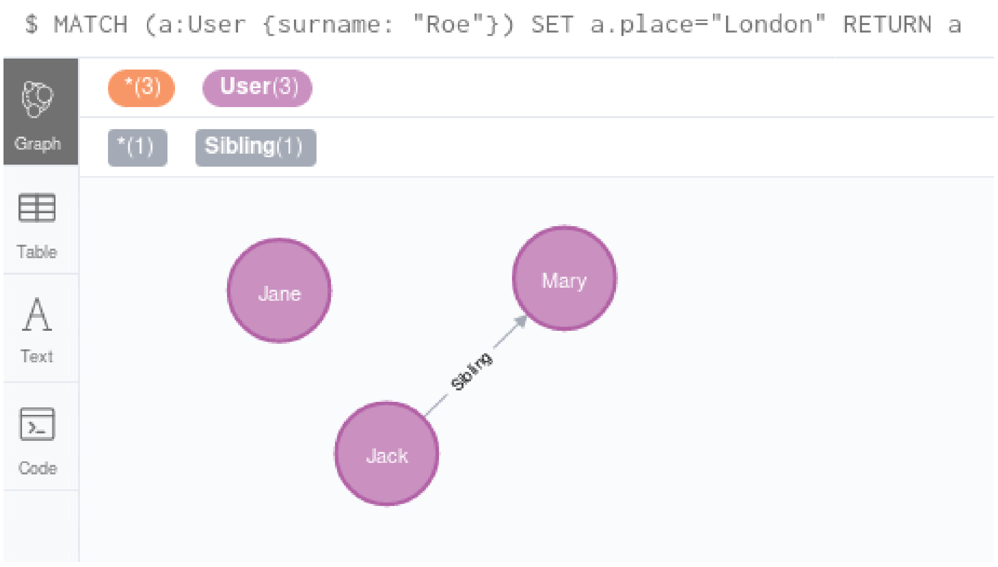
To set properties and labels you use the **SET** clause. Set age property for node Jack Roe:

```
$ MATCH (a:User {name:"Jack", surname:"Roe"}) SET a.age=34 RETURN a
```

Note that the SET clause here works on the nodes found using the MATCH clause. This means that we can set a property on a huge list of nodes if we don't write the MATCH clause carefully. The following query sets the city property on all the nodes with the surname property Roe:

```
$ MATCH (a:User {surname: "Roe"}) SET a.place="London" RETURN a
```

In our database, this query updates three nodes: `Jane, Jack, and Mary Roe.`



```
$ MATCH (a:User {surname: "Roe"}) SET a.place="London" RETURN a
```

You can change several assignment expressions to make more property changes at the same time. For example, to set the country as well, the query will be as follows:

```
$ MATCH (a:User {surname:"Roe"}) SET a.place="London",a.country="UK" RETURN a
```

The syntax to set a property to a relationship is the same.

```
$ MATCH (:User {surname:"Roe"})-[r:Knows]-() SET r.friend=TRUE
```

This command finds all the **`Knows`** relationships of users with the surname property Roe and sets the property **`friend`** to **`TRUE`** for all of them.

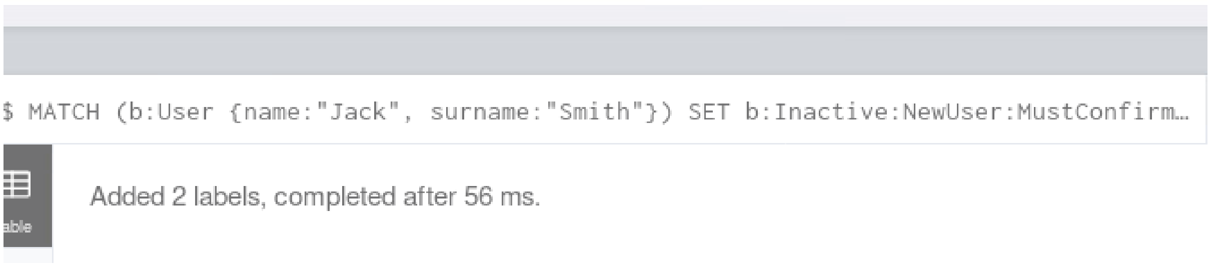
Lab 8.8 Adding labels to nodes.

The SET clause can also be used to add one or more labels to a node, as shown in the following query:

```
$ MATCH (b:User {name:"Jack", surname:"Smith"}) SET b:Inactive
```

The only difference is that we need to use the label separator instead of the property assignment. To chain more labels, just append them with the separator.

```
$ MATCH (b:User {name:"Jack", surname:"Smith"}) SET
  b:Inactive:NewUser:MustConfirmEmail
```

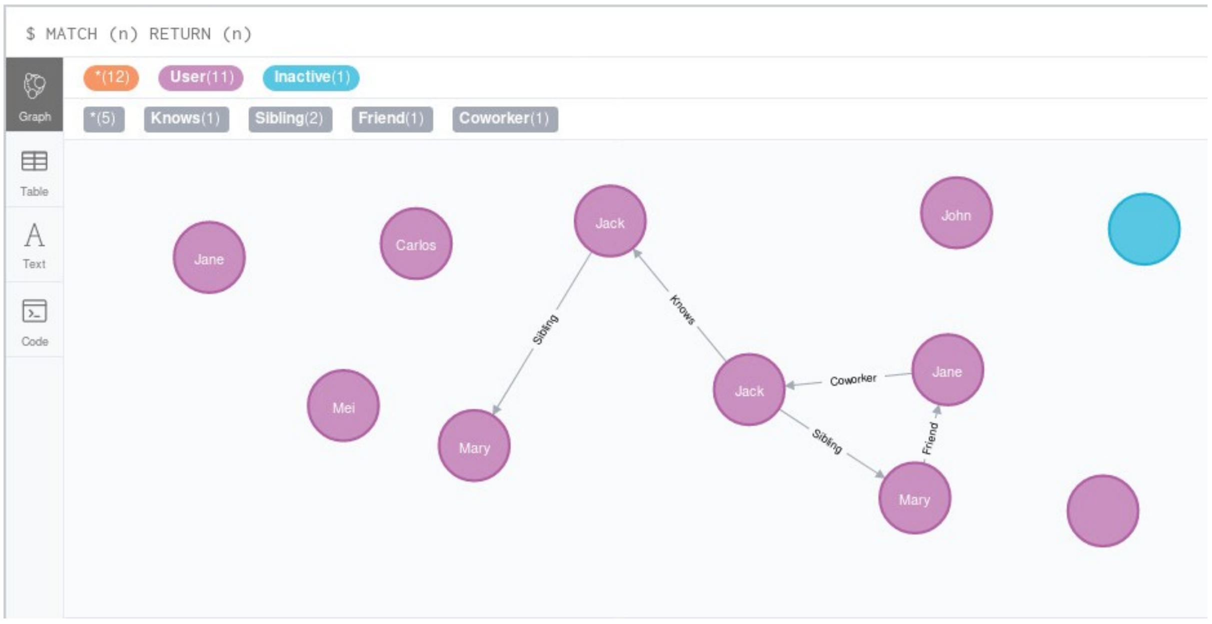


```
$ MATCH (b:User {name:"Jack", surname:"Smith"}) SET b:Inactive:NewUser:MustConfirm…
```

Added 2 labels, completed after 56 ms.

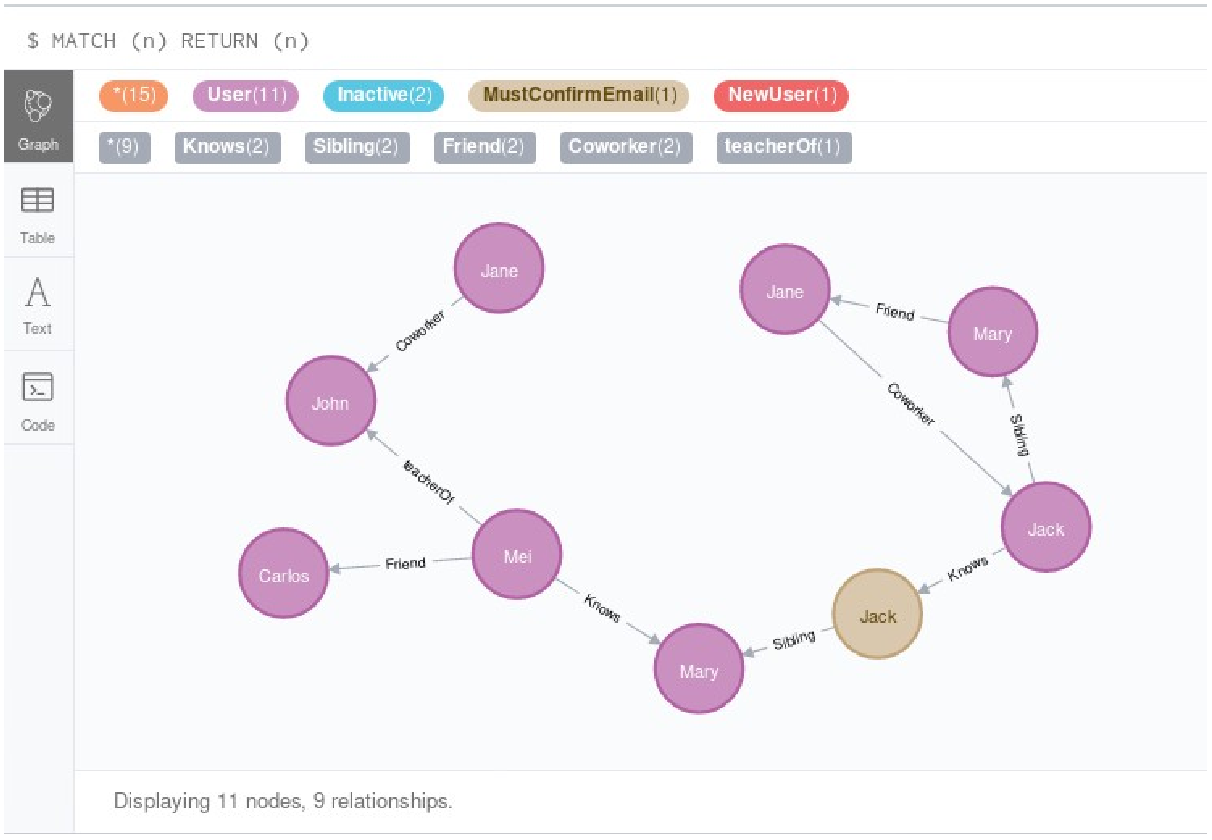
Lab 8.9 Complete your graph.

If you query the graph, you will see we have now a few nodes and a few relationships, but not all nodes have relationships between them.



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

Complete your graph by creating relationships of your choice between remaining nodes.



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

Displaying 11 nodes, 9 relationships.