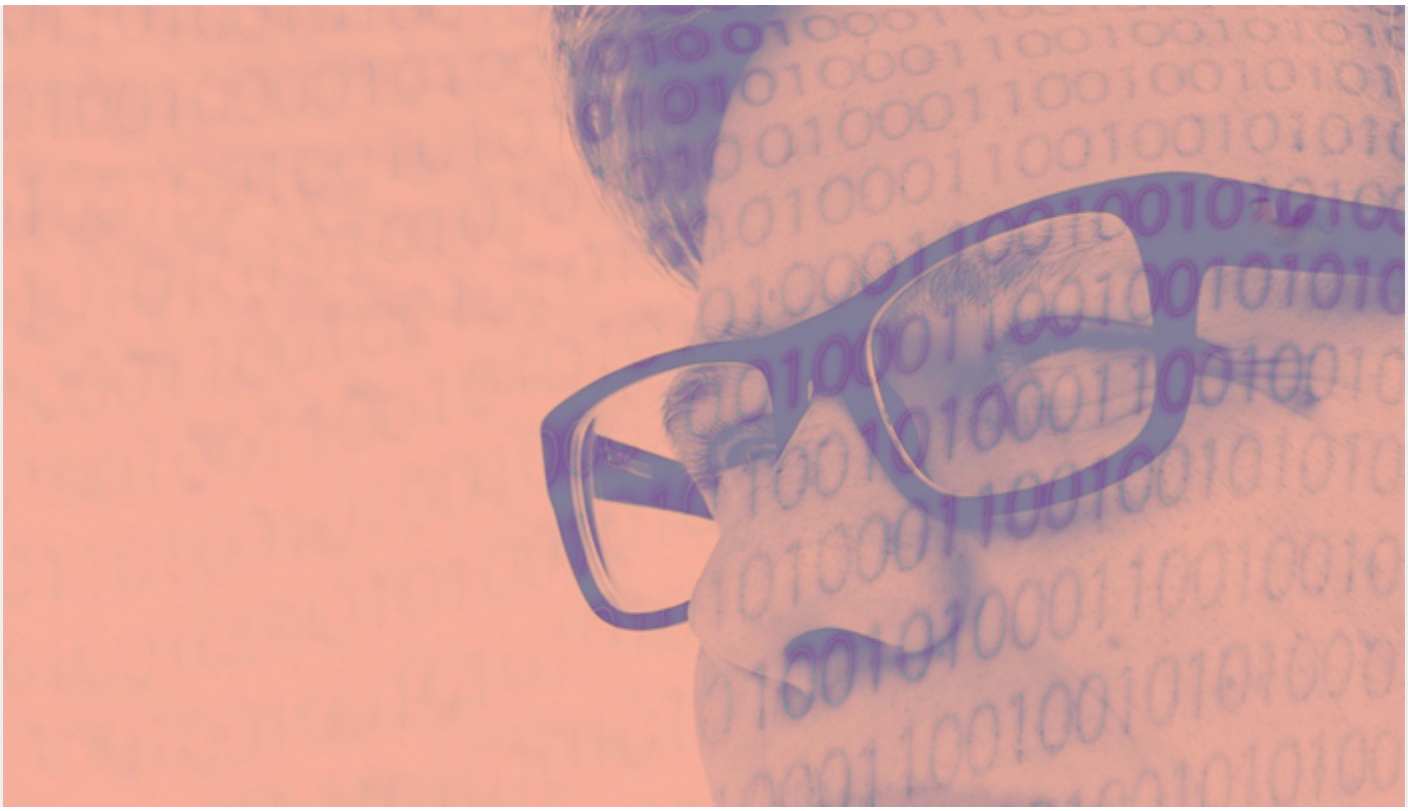
NEO4J GRAPH DATABASE QUERIES



In the previous blog post, where we introduced Neo4j. Here we explain queries.

First, start the server then open the shell:

```
cd neo4j bin

./neo4j start

./cypher-shell   -a bolt://localhost:7687  -u neo4j  -p xxx
```

(This article is part of our Data Visualization Guide. Use the right-hand menu to navigate.)

Create Node and Relations

Here we create **two nodes**. They each have label **Friends** and a **name** property. **Relations** both can have properties can have properties too.

```
CREATE (x:Friends { name: 'Walker' })
CREATE (y:Friends { name: 'Stephen' })
return x,y;

+-------------------------------------------------------------+
| x                           | y                             |
```

```
+-----------------------------------------------------------+
| (:Friends {name: "Walker"}) | (:Friends {name: "Stephen"}) |
+-----------------------------------------------------------+

1 row available after 15 ms, consumed after another 0 ms
Added 2 nodes, Set 2 properties, Added 2 labels
```

Note: Spelling and case is important. Neo4j give you no warning if anything is spelled wrong. So pay attention when creating, for example, relationships as they will be empty with no warning given. So look for **created x relationships** and **x rows available** after each statement to make sure it worked.

We then create a relationship (called an **edge**) from Stephen to Walker. We give this relation the arbitrary name **Friend**. We put letters **x**, **y**, and **r** in front of objects so that we can refer to them in subsequent steps. So it is shorthand notation. Plus it brings the created object into scope.

The procedure to create a relation is to bring the nodes together with a **MATCH** and **WHERE** statement, then issue the **CREATE** with a directional arrow, to indicate the direction of the relation, or a simple line, if it runs both ways. For example, if Stephen is a Friend of Walker then Walker is a Friend of Stephen. So the arrow could be a simple line.

You use double lines --, , when you want to indicate direction in queries (i.e, MATCH) statement, but not in the creation statement.

```
MATCH (x:Friends),(y:Friends)
WHERE y.name = "Stephen" AND x.name = "Walker"
CREATE (x)-->(y)
RETURN type(r);

+----------+
| type(r)  |
+----------+
| "Friend" |
+----------+
```

List all Friend Nodes

Now, let's try different queries. This one lists lists all nodes, since we did not give it any **WHERE** condition.

```
match(n:Friends)
       return n;
+------------------------------+
| n                            |
+------------------------------+
| (:Friends {name: "Walker"})  |
| (:Friends {name: "Stephen"}) |
+------------------------------+
```

List all **Friend** relations.

```
MATCH (a:Friends)-->(b:Friends)
       RETURN a.name, b.name;
+----------------------+
| a.name   | b.name    |
+----------------------+
| "Walker" | "Stephen" |
+----------------------+
```

Make Raj friends of Stephen.

```
CREATE (y:Friends { name: 'Raj' });

MATCH (x:Friends),(y:Friends)
WHERE y.name = "Stephen" AND x.name = "Raj"
CREATE (x)-->(y)
RETURN type(r);
```

Make sure the command(mainly the MATCH statement) worked by paying attention to the numbers returned. Remember what we said about spelling errors and case.

```
1 row available after 3 ms, consumed after another 0 ms
Created 1 relationships
```

Show all friends relationships. Note it now includes the one we just added

```
MATCH (a:Friends)-->(b:Friends)
       RETURN a.name, b.name;
+----------------------+
| a.name   | b.name    |
+----------------------+
| "Walker" | "Stephen" |
| "Raj"    | "Stephen" |
+----------------------+
```

Add 1 more, Teresa, then make her a Friend with Raj.

```
CREATE (x:Friends { name: 'Teresa' })
return x;

MATCH (x:Friends),(y:Friends)
WHERE y.name = "Stephen" AND x.name = "Raj"
CREATE (x)-->(y);

MATCH (x:Friends),(y:Friends)
WHERE y.name = "Raj" AND x.name = "Teresa"
CREATE (x)-->(y)
RETURN type(r);
```

Below we shown are friends of friends who are not of each other. That would be Teresa and Stephen are not friends.

To put this in terms of mathematics, which might make it simpler to understand, we could say that we look for the intersection of the set **u** and **notFriend** and then applying WHERE NOT that find objects outside that intersection.

To recall what friends we have to make this easier to see, first:

```
Match (a:Friends)--(b)
       return a,b;
+-------------------------------------------------------------+
| a                            | b                            |
+-------------------------------------------------------------+
| (:Friends {name: "Walker"})  | (:Friends {name: "Stephen"}) |
| (:Friends {name: "Stephen"}) | (:Friends {name: "Raj"})     |
| (:Friends {name: "Stephen"}) | (:Friends {name: "Raj"})     |
| (:Friends {name: "Stephen"}) | (:Friends {name: "Walker"})  |
| (:Friends {name: "Raj"})     | (:Friends {name: "Teresa"})  |
| (:Friends {name: "Raj"})     | (:Friends {name: "Stephen"}) |
| (:Friends {name: "Raj"})     | (:Friends {name: "Stephen"}) |
| (:Friends {name: "Teresa"})  | (:Friends {name: "Raj"})     |
+-------------------------------------------------------------+
```

The friends who.are-not-friends-of query lists Teresa and Stephen because Teresa is not a friend of Stephen. In other words, the first part of the query shows all relations for those people listed in variable **u** then it lists **who is not in u**.

```
Match (u:Friends)-->(:Friends)-->(notFriend:Friends)
       WHERE NOT (u)-->(notFriend)
       RETURN u, notFriend;
+-------------------------------------------------------------+
| u                            | notFriend                    |
+-------------------------------------------------------------+
| (:Friends {name: "Teresa"}) | (:Friends {name: "Stephen"}) |
| (:Friends {name: "Teresa"}) | (:Friends {name: "Stephen"}) |
+-------------------------------------------------------------+
```

Directed Relations

Now we create a relationship in the opposite direction **Friend A < - Friend B** instead of **Friend A -> Friend B** by putting the directional arrow in the first position in the CREATE statement.

```
MATCH (x:Friends),(y:Friends)
WHERE y.name = "Raj" AND x.name = "Teresa"
CREATE (x)< --(y)
RETURN type(r);
```

Now list all relations, first the left-hand then non-directional then going rightward. An outbound relationship from a Friends node to another is the same as an inbound relationship since the nodes are the same. So the first and last queries list the same 4 relations. And a bi-directional query lists 8 since an inbound and an outbound both qualify as a relation in either direction, which is what the --

means..

It would be simpler to see all of this if we had called one set of nodes Customers and another AccountManager, since there will be no logical symmetry. We will build up examples like that is subsequent blog posts.

```
Match (a:Friends)< --(b) return a,b; +-----------------------------------
---------------------+ | a | b | +-----------------------------------------
-----------------+ | (:Friends {name: "Stephen"}) | (:Friends {name: "Raj"})
| | (:Friends {name: "Stephen"}) | (:Friends {name: "Raj"}) | | (:Friends
{name: "Stephen"}) | (:Friends {name: "Walker"}) | | (:Friends {name: "Raj"})
| (:Friends {name: "Teresa"}) | +-----------------------------------------
----------------+ Match (a:Friends)--(b) return a,b; +---------------------
-----------------------------------+ | a | b | +-------------------------
---------------------------------+ | (:Friends {name: "Walker"}) |
(:Friends {name: "Stephen"}) | | (:Friends {name: "Stephen"}) | (:Friends
{name: "Raj"}) | | (:Friends {name: "Stephen"}) | (:Friends {name: "Raj"}) |
| (:Friends {name: "Stephen"}) | (:Friends {name: "Walker"}) | | (:Friends
{name: "Raj"}) | (:Friends {name: "Teresa"}) | | (:Friends {name: "Raj"}) |
(:Friends {name: "Stephen"}) | | (:Friends {name: "Raj"}) | (:Friends {name:
"Stephen"}) | | (:Friends {name: "Teresa"}) | (:Friends {name: "Raj"}) | +---
-----------------------------------------------------------+ 8 rows available
after 22 ms, consumed after another 2 ms Match (a:Friends)-->(b)
       return a,b;
+-------------------------------------------------------------+
| a                           | b                             |
+-------------------------------------------------------------+
| (:Friends {name: "Walker"}) | (:Friends {name: "Stephen"}) |
| (:Friends {name: "Raj"})    | (:Friends {name: "Stephen"}) |
| (:Friends {name: "Raj"})    | (:Friends {name: "Stephen"}) |
| (:Friends {name: "Teresa"}) | (:Friends {name: "Raj"})     |
+-------------------------------------------------------------+

4 rows available after 24 ms, consumed after another 2 ms
```