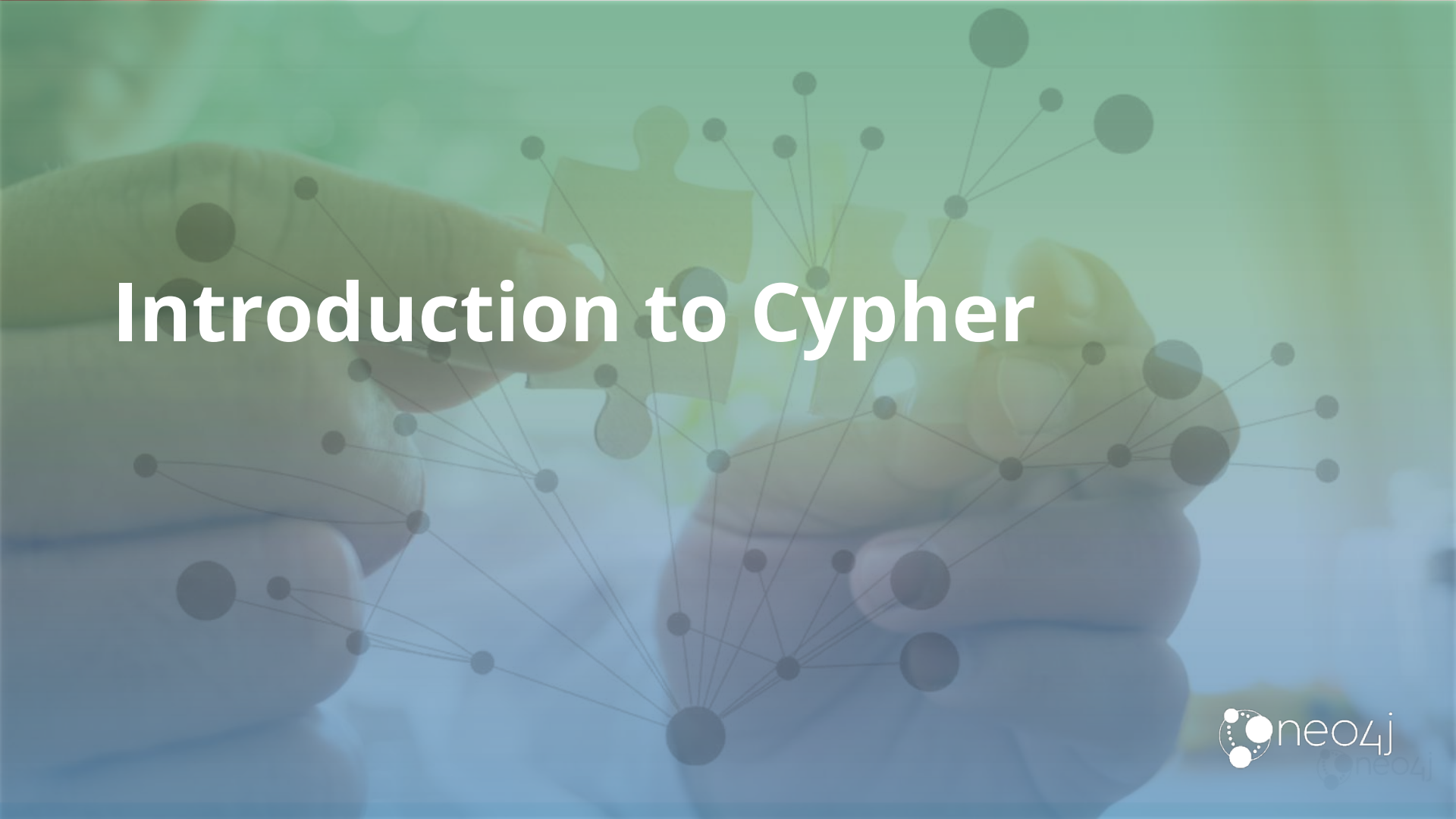
Introduction to Cypher



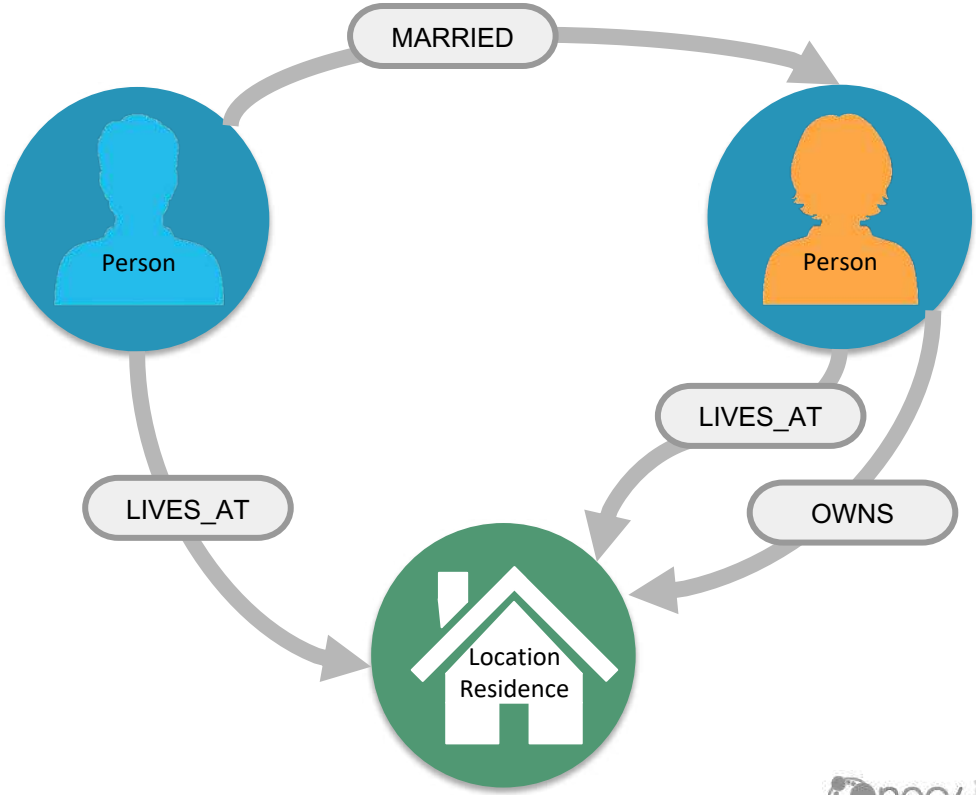
Overview

At the end of this module, you should be able to write Cypher statements to:

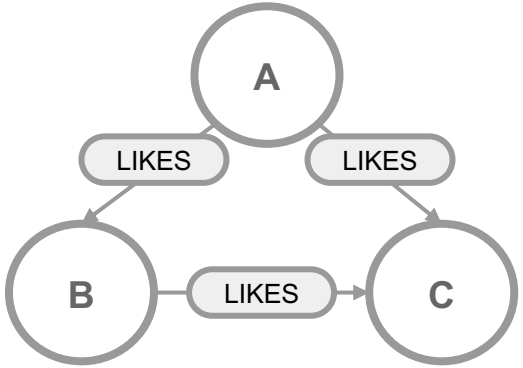
- Retrieve nodes from the graph.
- Filter nodes retrieved using labels and property values of nodes.
- Retrieve property values from nodes in the graph.
- Filter nodes retrieved using relationships.

What is Cypher?

- Declarative query language
- Focuses on **what**, not how to retrieve
- Uses keywords such as **MATCH**, **WHERE**, **CREATE**
- Runs in the database server for the graph
- ASCII art to represent nodes and relationships



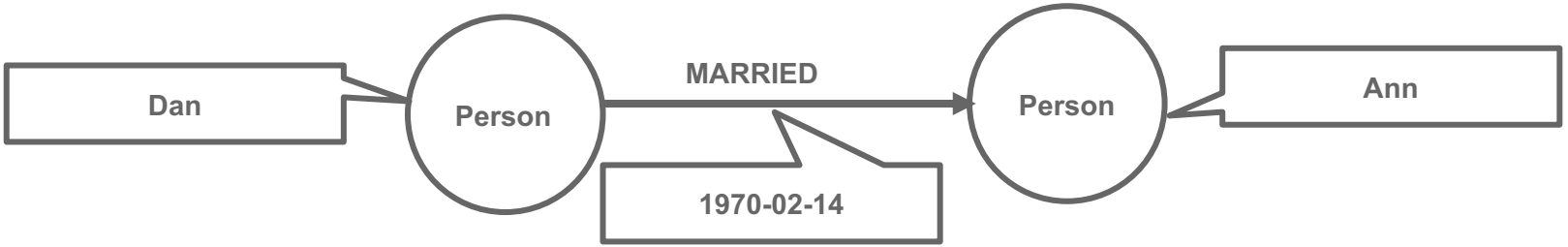
Cypher is ASCII Art

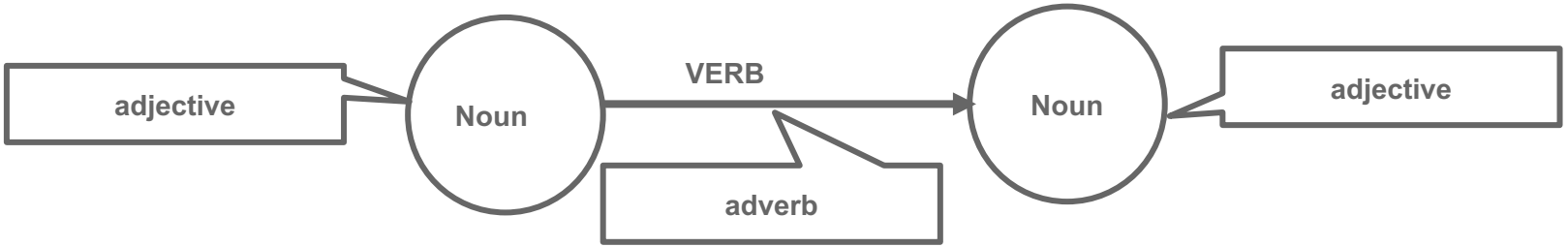



```
(A)-[:LIKES]->(B),(A)-[:LIKES]->(C),(B)-[:LIKES]->(C)

(A)-[:LIKES]->(B)-[:LIKES]->(C)<-[:LIKES]-(A)
```

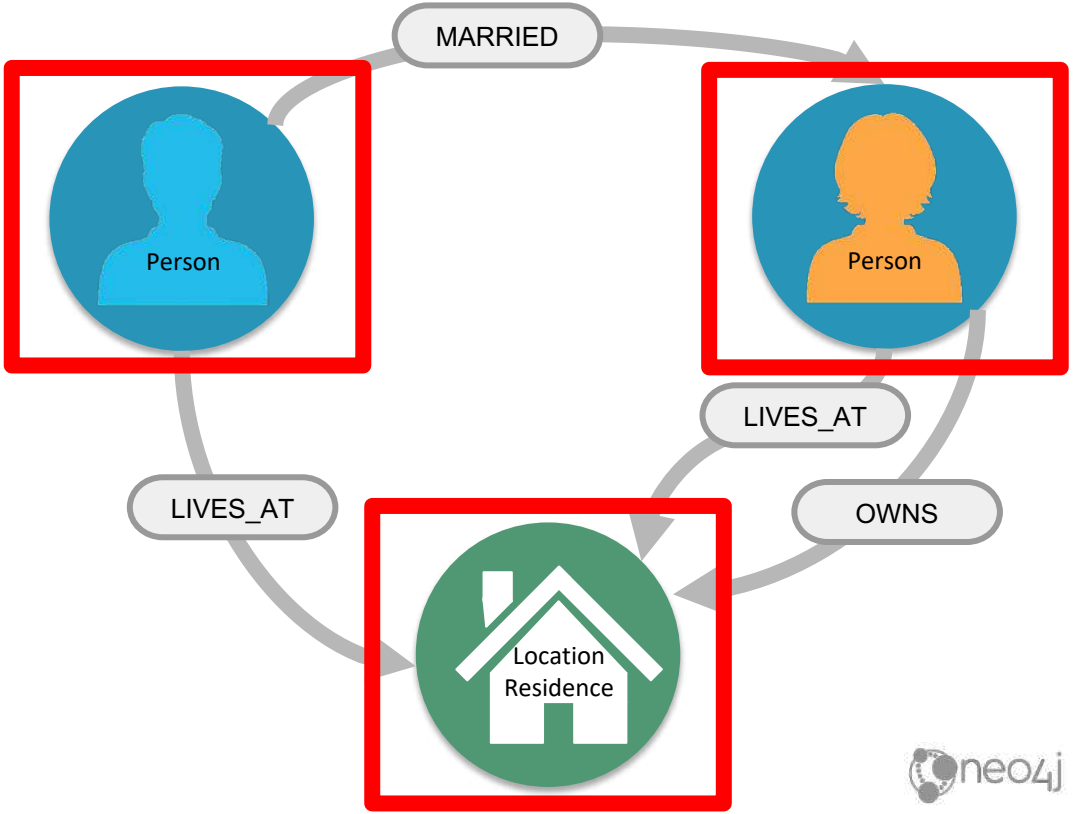
Cypher is readable





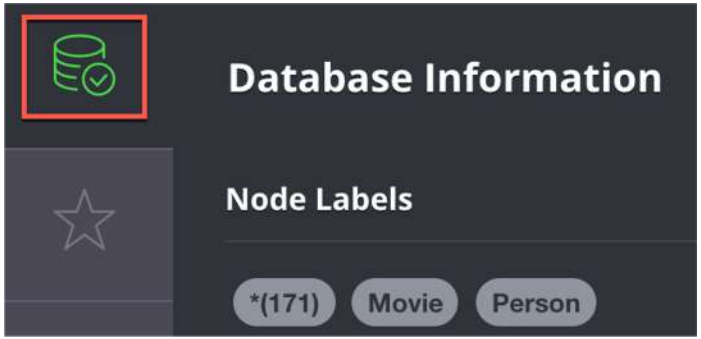
Nodes

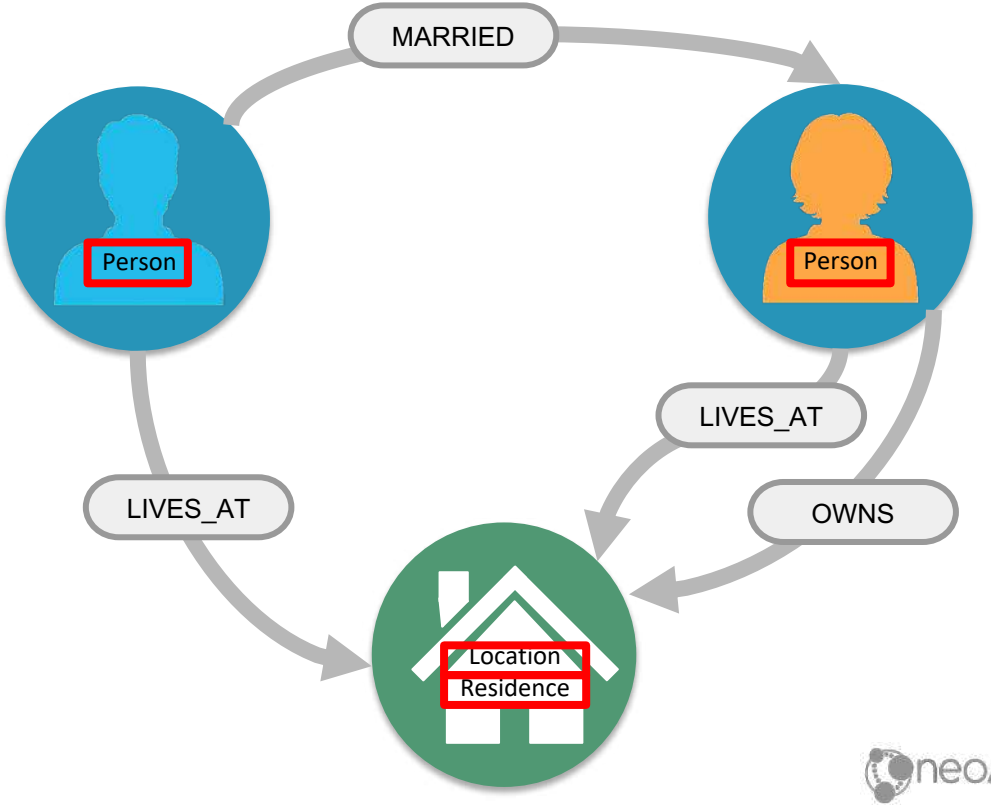
```
()
(p)
(l)
(n)
```


MARRIED
Person
Person
LIVES_AT
LIVES_AT
OWNS
Location
Residence

Labels

```
(:Person)
(p:Person)
(:Location)
(l:Location)
(n:Residence)
(x:Location:Residence)
```

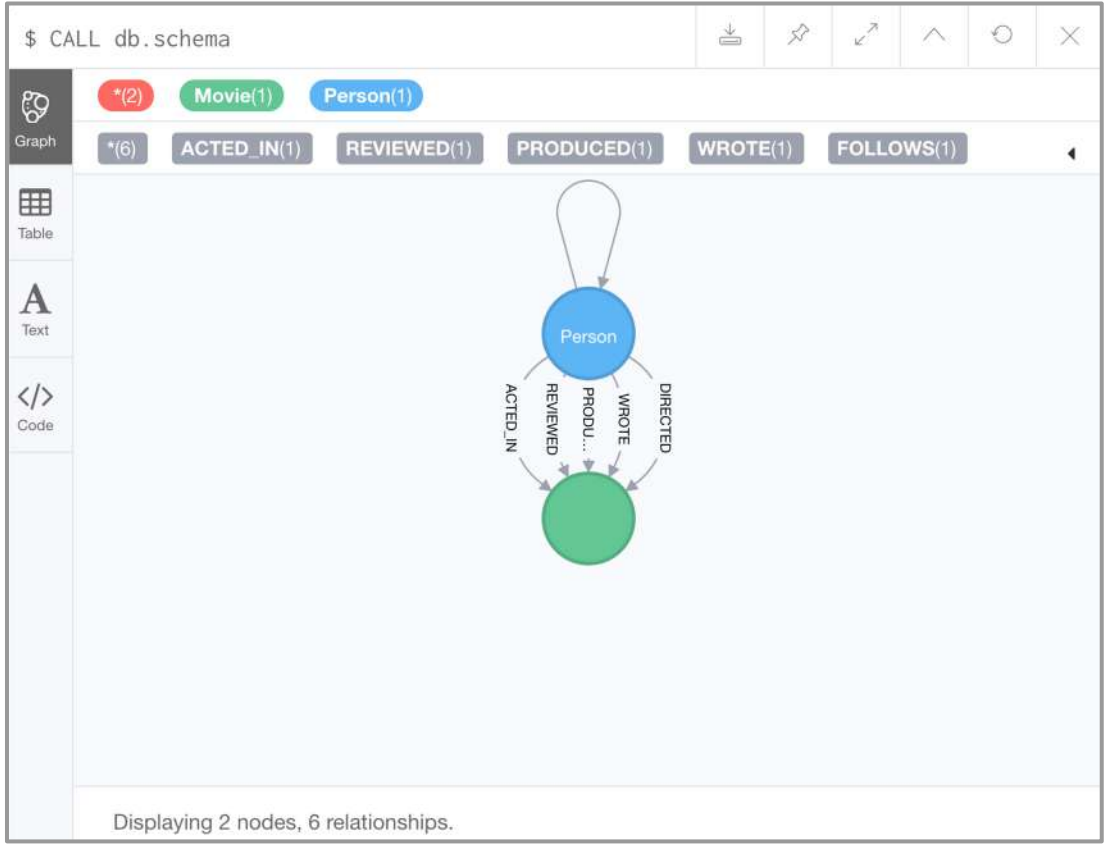




Comments in Cypher

```
()                    // anonymous node not be referenced later in the query
(p)                   // variable p, a reference to a node used later
(:Person)             // anonymous node of type Person
(p:Person)            // p, a reference to a node of type Person
(p:Actor:Director)    // p, a reference to a node of types Actor and Director
```

Examining the data model

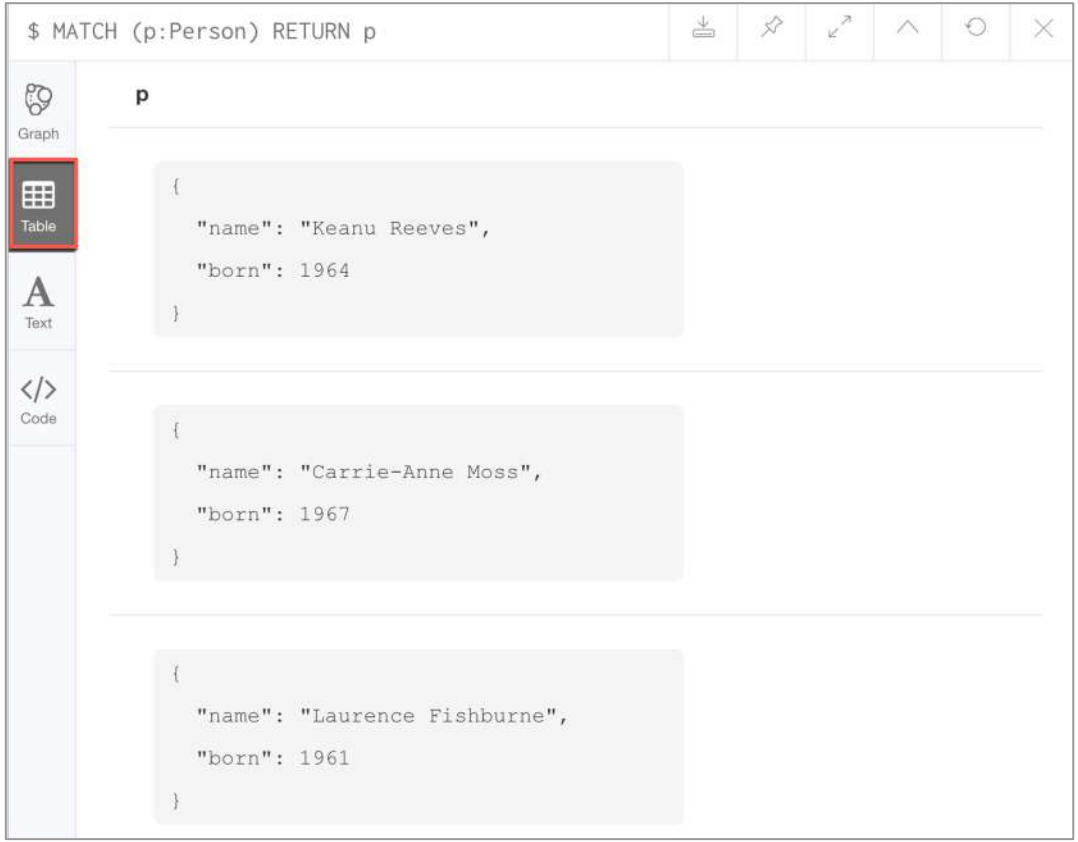


Using MATCH to retrieve nodes

```
MATCH (n) // returns all nodes in the graph
RETURN n
```

```
MATCH (p:Person) // returns all Person nodes in the graph
RETURN p
```

Viewing nodes as table data



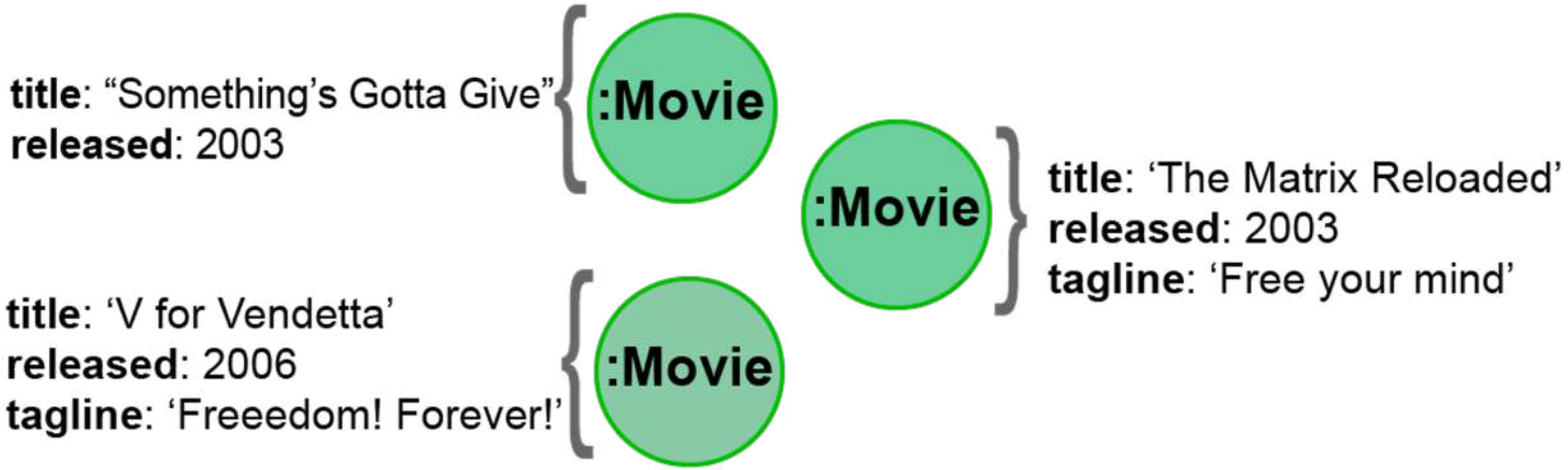
Exercise 1: Retrieving Nodes

In Neo4j Browser:

:play intro-exercises

Then follow instructions for Exercise 1.

Properties


title: “Something’s Gotta Give”
released: 2003
:Movie
:Movie
title: ‘The Matrix Reloaded’
released: 2003
tagline: ‘Free your mind’
title: ‘V for Vendetta’
released: 2006
tagline: ‘Freeedom! Forever!’
:Movie

Examining property keys

```
CALL db.propertyKeys
```

```
$ CALL db.propertyKeys
```

Table | Text | Code

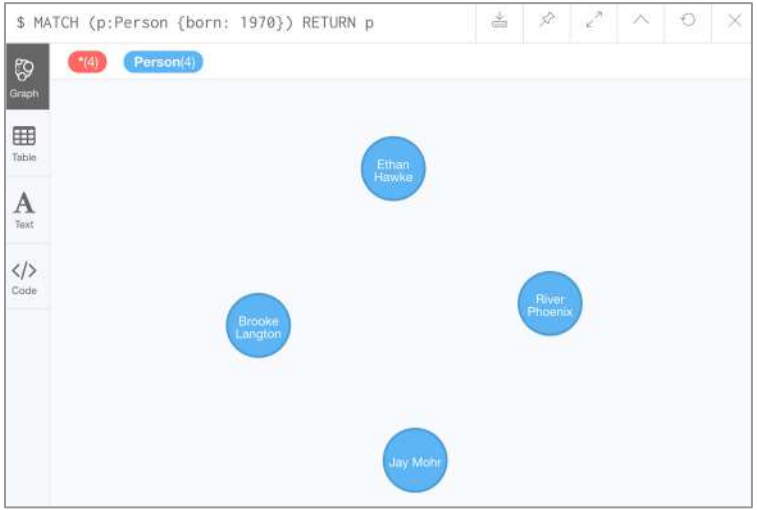
propertyKey
"title"
"released"
"tagline"
"name"
"born"
"roles"
"summary"
"rating"
"id"
"share_link"
"favorite_count"
"display_name"

Started streaming 12 records in less than 1 ms and completed in less than 1 ms.

Retrieving nodes filtered by a property value - 1

Find all *people* born in *1970, returning the nodes*:

```
MATCH (p:Person {born: 1970})
RETURN p
```



Retrieving nodes filtered by a property value - 2

Find all movies released in *2003* with the tagline,

Free your mind, returning the nodes:

```
MATCH (m:Movie {released: 2003, tagline: 'Free your mind'})
RETURN m
```

Returning property values

Find all people born in *1965* and return their names:

```
MATCH (p:Person {born: 1965})
RETURN p.name, p.born
```

Specifying aliases

```
MATCH (p:Person {born: 1965})
RETURN p.name AS name, p.born AS `birth year`
```

```
$ MATCH (p:Person {born: 1965}) RETURN p.name AS name, p.born AS `birth year`
```

Table | Text | Code

name	birth year
"Lana Wachowski"	1965
"Tom Tykwer"	1965
"John C. Reilly"	1965

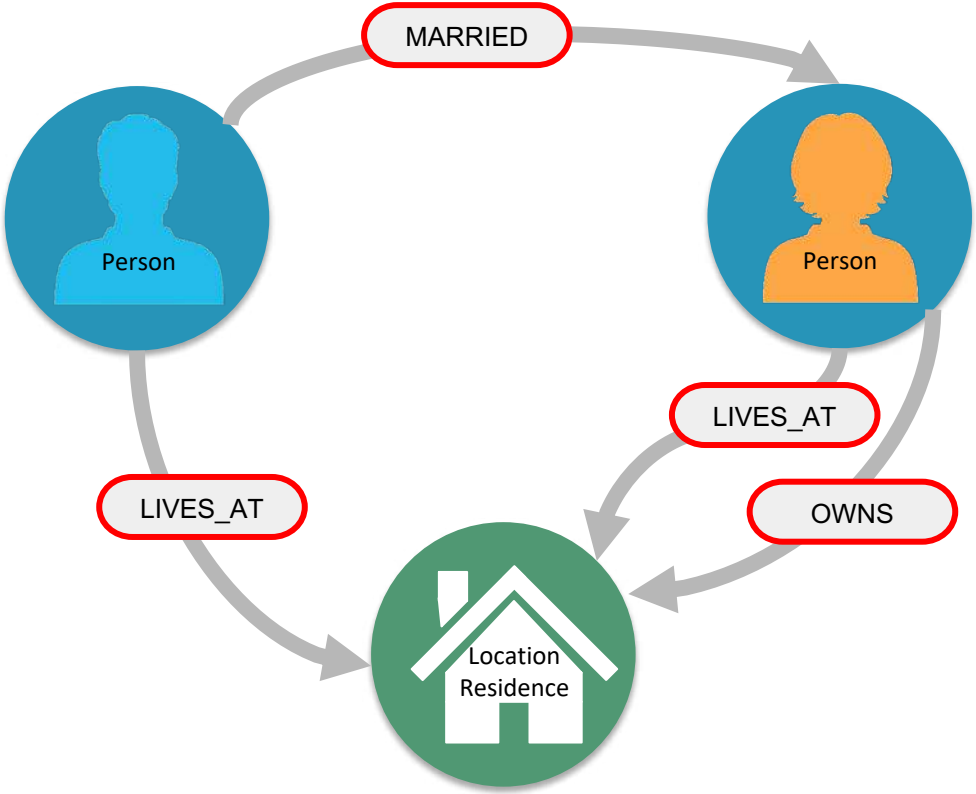
Exercise 2: Filtering queries using property values

In Neo4j Browser:

:play intro-exercises

Then follow instructions for Exercise 2.

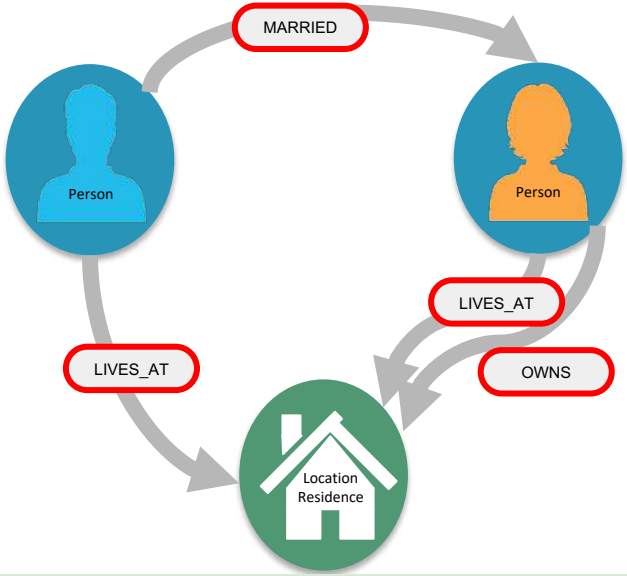
Relationships



ASCII art for nodes and relationships

```
()                          // a node
()--()      // 2 nodes have some type of relationship
()-->()     // the first node has a relationship to the second node
()<--()     // the second node has a relationship to the first node
```

Querying using relationships

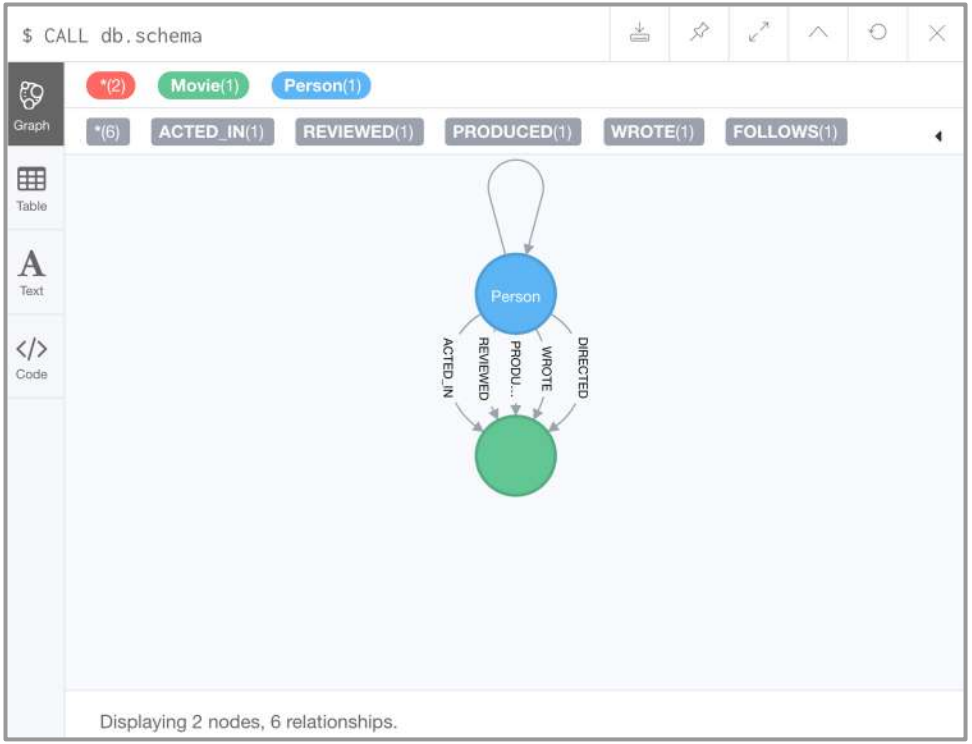



```
MATCH (p:Person)-[:LIVES_AT]->(h:Residence)
RETURN p.name, h.address
```

```
MATCH (p:Person)--(h:Residence) // any relationship
RETURN p.name, h.address
```

Examining relationships

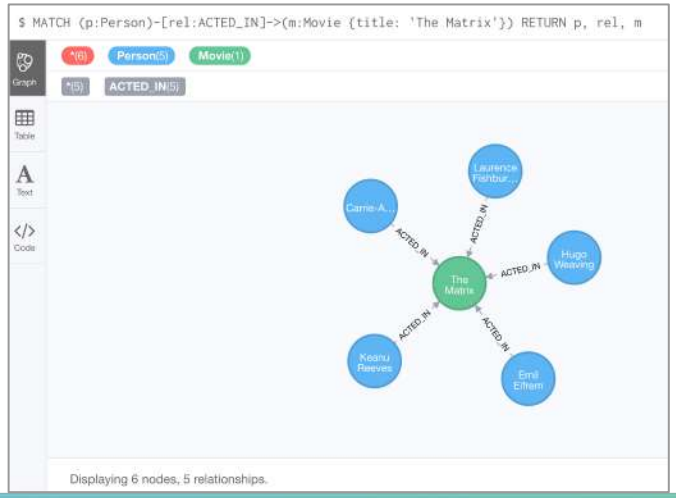
```
CALL db.schema.visualization()
```



Using a relationship in a query

Find all people who acted in the movie, *The Matrix*, returning the nodes and relationships found:

```
MATCH (p:Person)-[rel:ACTED_IN]->(m:Movie {title: 'The Matrix'})
RETURN p, rel, m
```



Querying by multiple relationships

Find all movies that *Tom Hanks* acted in or directed and return the title of the move:

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

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

Table | Text | Code

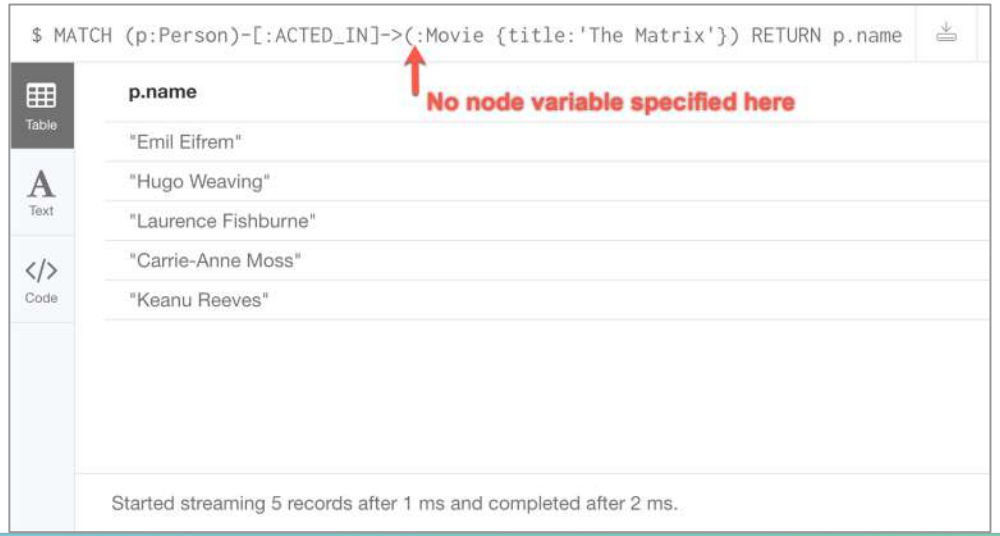
p.name	m.title
"Tom Hanks"	"Apollo 13"
"Tom Hanks"	"Cast Away"
"Tom Hanks"	"The Polar Express"
"Tom Hanks"	"A League of Their Own"
"Tom Hanks"	"Charlie Wilson's War"
"Tom Hanks"	"Cloud Atlas"
"Tom Hanks"	"The Da Vinci Code"
"Tom Hanks"	"The Green Mile"
"Tom Hanks"	"You've Got Mail"
"Tom Hanks"	"That Thing You Do"
"Tom Hanks"	"That Thing You Do"
"Tom Hanks"	"Joe Versus the Volcano"
"Tom Hanks"	"Sleepless in Seattle"

Started streaming 13 records after 1 ms and completed after 1 ms.

Using anonymous nodes in a query

Find all people who acted in the movie, *The Matrix* and return their names:

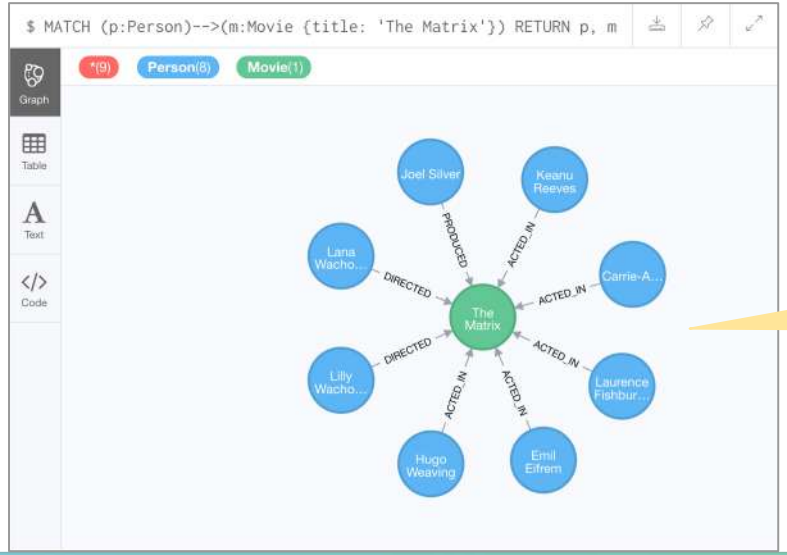
```
MATCH (p:Person)-[:ACTED_IN]->(:Movie {title: 'The Matrix'})
RETURN p.name
```



Using an anonymous relationship for a query

Find all people who have any type of relationship to the movie, *The Matrix* and return the nodes:

```
MATCH (p:Person)-->(m:Movie {title: 'The Matrix'})
RETURN p, m
```



Connect result nodes enabled in Neo4j Browser

Retrieving relationship types

Find all people who have any type of relationship to the movie, *The Matrix* and return the name of the person and their relationship type:

```
MATCH (p:Person)-[rel]->(:Movie {title:'The Matrix'})
RETURN p.name, type(rel)
```

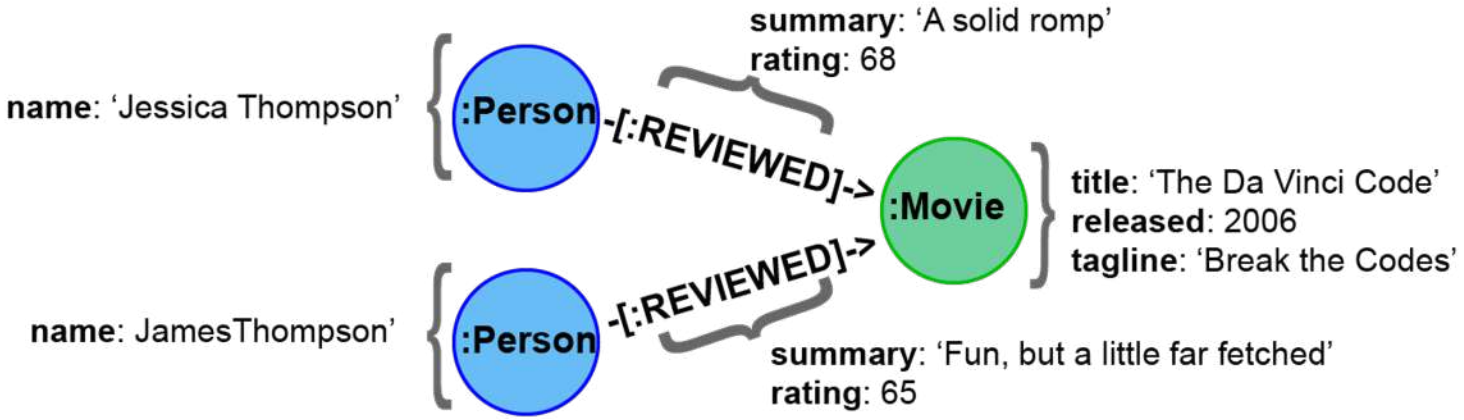
```
$ MATCH (p:Person)-[rel]->(:Movie {title:'The Matrix'}) RETURN p.name, type(rel)
```

Table | Text | Code

p.name	type(rel)
"Emil Eifrem"	"ACTED_IN"
"Joel Silver"	"PRODUCED"
"Lana Wachowski"	"DIRECTED"
"Lilly Wachowski"	"DIRECTED"
"Hugo Weaving"	"ACTED_IN"
"Laurence Fishburne"	"ACTED_IN"
"Carrie-Anne Moss"	"ACTED_IN"
"Keanu Reeves"	"ACTED_IN"

Started streaming 8 records after 1 ms and completed after 2 ms.

Retrieving properties for a relationship - 1



Retrieving properties for a relationship - 2

Find all people who gave the movie, *The Da Vinci Code*, a rating of *65*, returning their names:

```
MATCH (p:Person)-[:REVIEWED {rating: 65}]->(:Movie {title: 'The Da Vinci Code'})
RETURN p.name
```

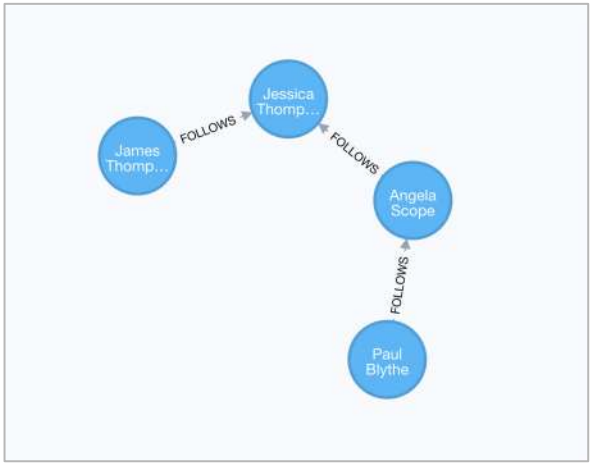
```
$ MATCH (p:Person)-[:REVIEWED {rating: 65}]->(:Movie {title: 'The Da Vinci Code'}) RETURN p.name
```

Table

Text

p.name
"James Thompson"

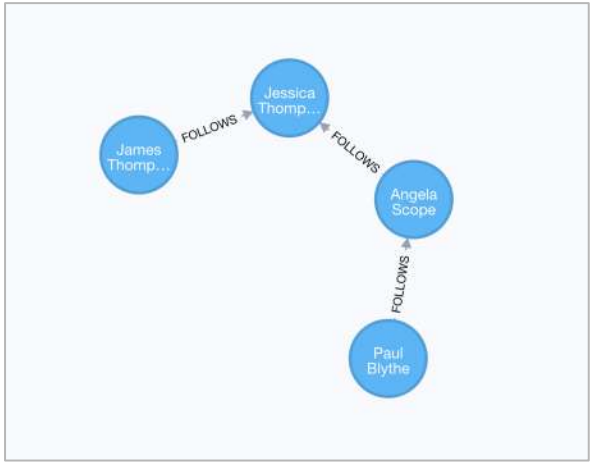
Using patterns for queries - 1



Find all people who follow *Angela Scope*, returning the nodes:

```
MATCH  (p:Person)-[:FOLLOWS]->(:Person {name:'Angela
Scope'})
RETURN p
```

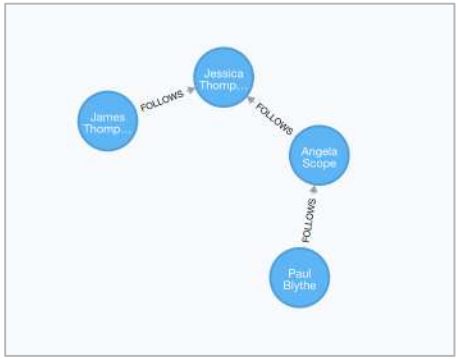
Using patterns for queries - 2



Find all people who *Angela Scope* follows, returning the nodes:

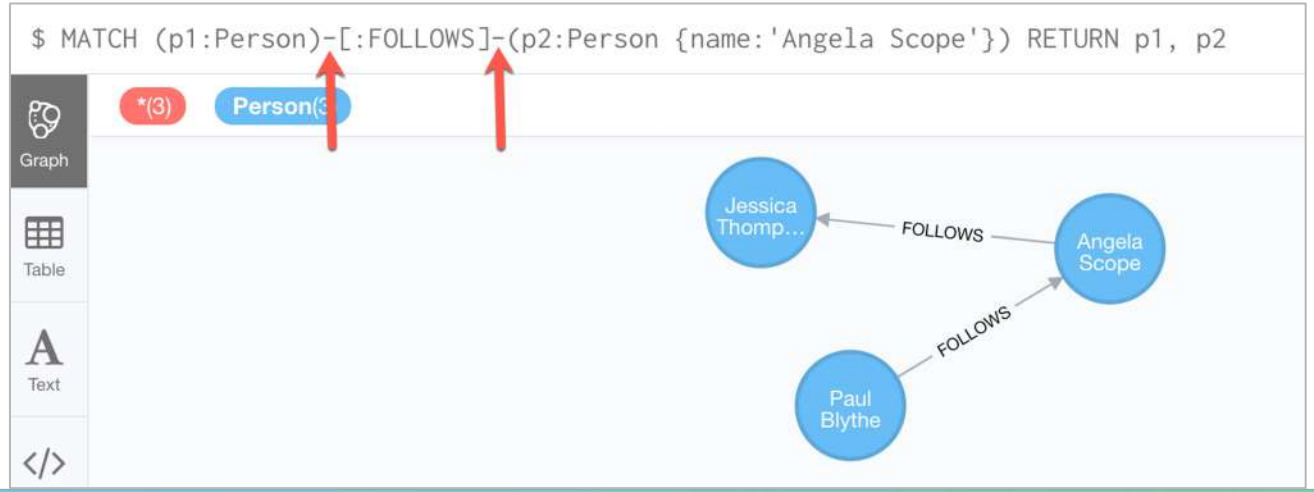
```
MATCH  (p:Person)<-[:FOLLOWS]-(:Person {name:'Angela
Scope'})
RETURN p
```

Querying by any direction of the relationship

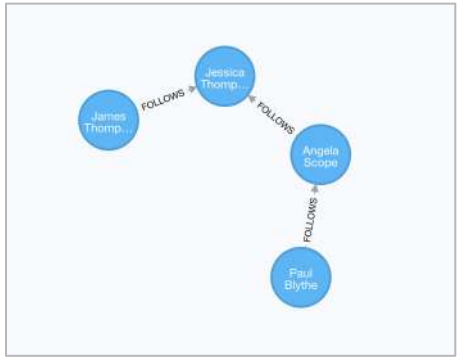


Find all people who follow *Angela Scope* or who *Angela Scope* follows, returning the nodes:

```
MATCH  (p1:Person)-[:FOLLOWS]-(p2:Person {name:'Angela Scope'})
RETURN p1, p2
```



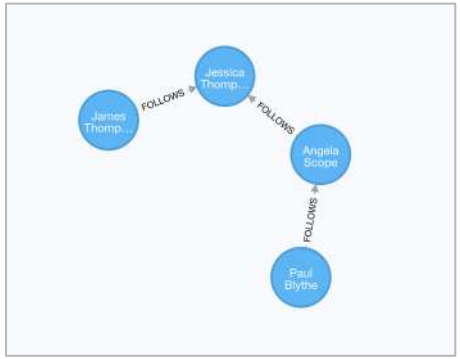
Traversing relationships - 1



Find all people who follow anybody who follows *Jessica Thompson* returning the people as nodes:

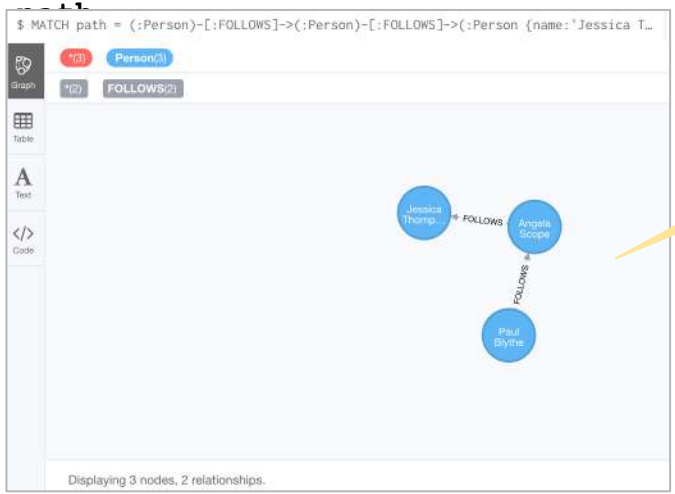
```
MATCH  (p:Person)-[:FOLLOWS]->(:Person)-[:FOLLOWS]->
       (:Person {name:'Jessica Thompson'})
RETURN p
```

Traversing relationships - 2



Find the path that includes all people who follow anybody who follows *Jessica Thompson* returning the path:

```
MATCH  path = (:Person)-[:FOLLOWS]->(:Person)-[:FOLLOWS]->
                (:Person {name:'Jessica Thompson'})
RETURN path
```

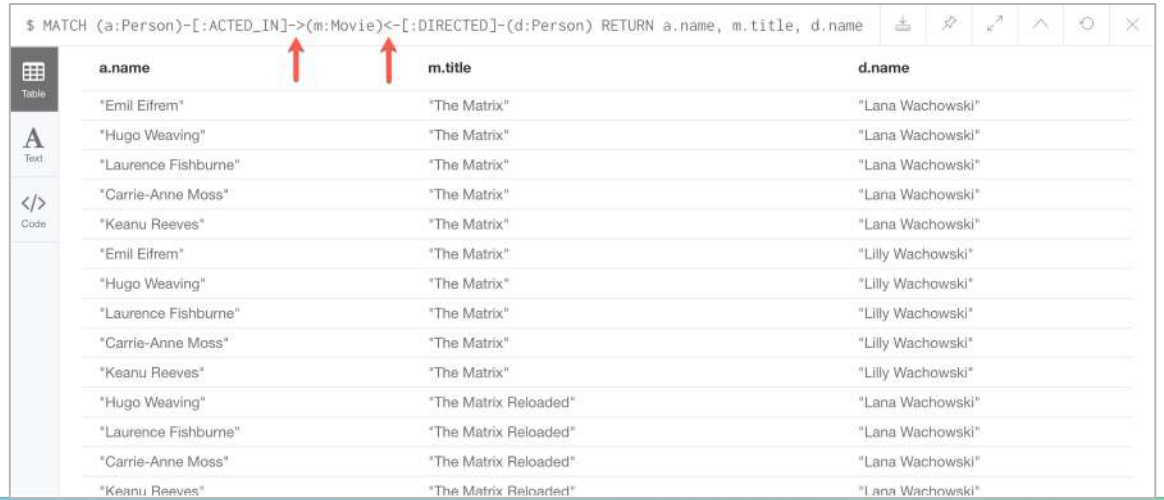


Sub-graph

Using relationship direction to optimize a query

Find all people that acted in a movie and the directors for that same movie, returning the name of the actor, the movie title, and the name of the director:

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)<-[:DIRECTED]-(d:Person)
RETURN a.name, m.title, d.name
```



a.name	m.title	d.name
"Emil Eifrem"	"The Matrix"	"Lana Wachowski"
"Hugo Weaving"	"The Matrix"	"Lana Wachowski"
"Laurence Fishburne"	"The Matrix"	"Lana Wachowski"
"Carrie-Anne Moss"	"The Matrix"	"Lana Wachowski"
"Keanu Reeves"	"The Matrix"	"Lana Wachowski"
"Emil Eifrem"	"The Matrix"	"Lilly Wachowski"
"Hugo Weaving"	"The Matrix"	"Lilly Wachowski"
"Laurence Fishburne"	"The Matrix"	"Lilly Wachowski"
"Carrie-Anne Moss"	"The Matrix"	"Lilly Wachowski"
"Keanu Reeves"	"The Matrix"	"Lilly Wachowski"
"Hugo Weaving"	"The Matrix Reloaded"	"Lana Wachowski"
"Laurence Fishburne"	"The Matrix Reloaded"	"Lana Wachowski"
"Carrie-Anne Moss"	"The Matrix Reloaded"	"Lana Wachowski"
"Keanu Reeves"	"The Matrix Reloaded"	"Lana Wachowski"

Cypher style recommendations - 1

Here are the **Neo4j-recommended** Cypher coding standards that we use in this training:

- Node labels are CamelCase and case-sensitive (examples: *Person*, *NetworkAddress*).
- Property keys, variables, parameters, aliases, and functions are camelCase case-sensitive (examples: *businessAddress*, *title*).
- Relationship types are in upper-case and can use the underscore. (examples: *ACTED_IN*, *FOLLOWS*).
- Cypher keywords are upper-case (examples: MATCH, RETURN).

Exercise 3: Filtering queries using relationships

In Neo4j Browser:

:play intro-exercises

Then follow instructions for Exercise 3.

Summary

You should be able to write Cypher statements to:

- Retrieve nodes from the graph.
- Filter nodes retrieved using labels and property values of nodes.
- Retrieve property values from nodes in the graph.
- Filter nodes retrieved using relationships.