Exercise #2: Introduction to NoSQL (graph databases)
Over the past years, graph databases have become increasingly popular. Many large companies like Facebook, LinkedIn and Twitter use them heavily for their social media networks. However, when selecting a database solution for a project, this category of NoSQL databases is often overlooked in favor of the traditional relational database or RDBMS for short.

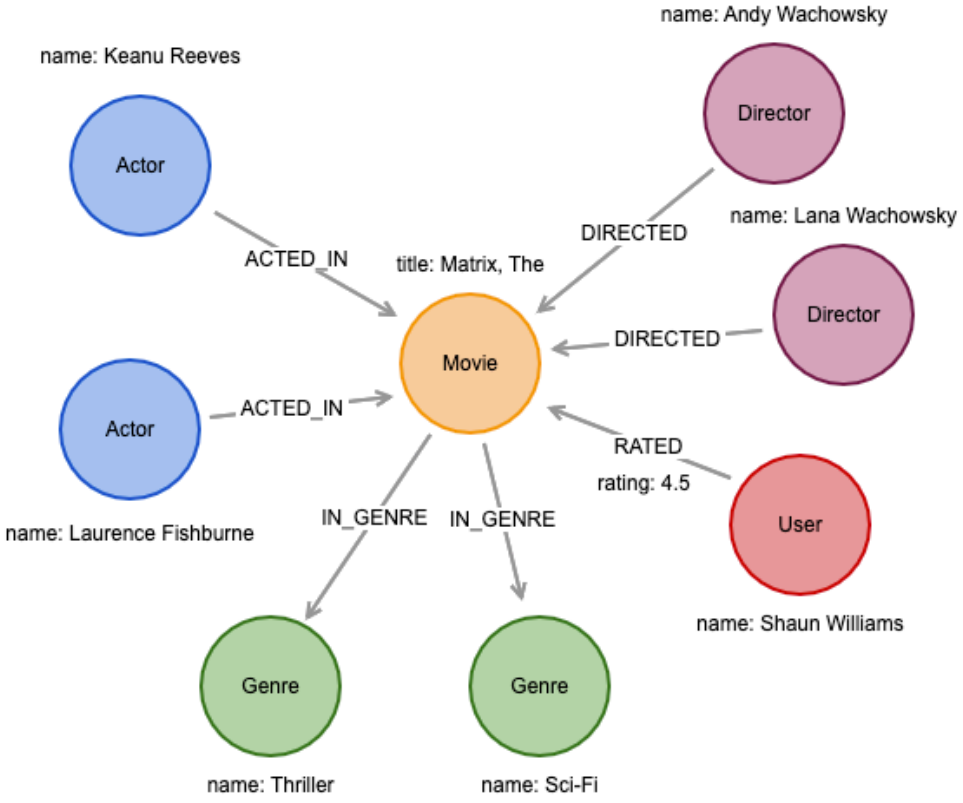
Today's goal is to get a basic understanding of graph NoSQL databases by learning about Neo4J, which is the most popular graph database solution to date. It's open source and written in Java. **It provides a free online sandbox (so, no need to install anything).**

In Neo4j, data is stored as a directed graph. A graph consists of nodes and the relationships between these nodes. Nodes are labelled to denounce their type. When compared to an RDBMS, you can think of nodes and labels as records and tables.

Relationships connect nodes. In Neo4j, they are first class citizens and can be queried directly. There is no need to compute them at runtime as an RDBMS will do when joining tables. Each relationship between two nodes has an orientation, hence the term directed graph.

Finally, nodes and relationships both support properties in the form of key value pairs.

Graphs can be easily visualized as shown in the figure below. It represents sample data from a movie graph database. It consists of **nodes** labelled **Actor, Director, Movie, Genre** and **User. Relationships** are drawn as arrows pointing in the direction of the relationship.

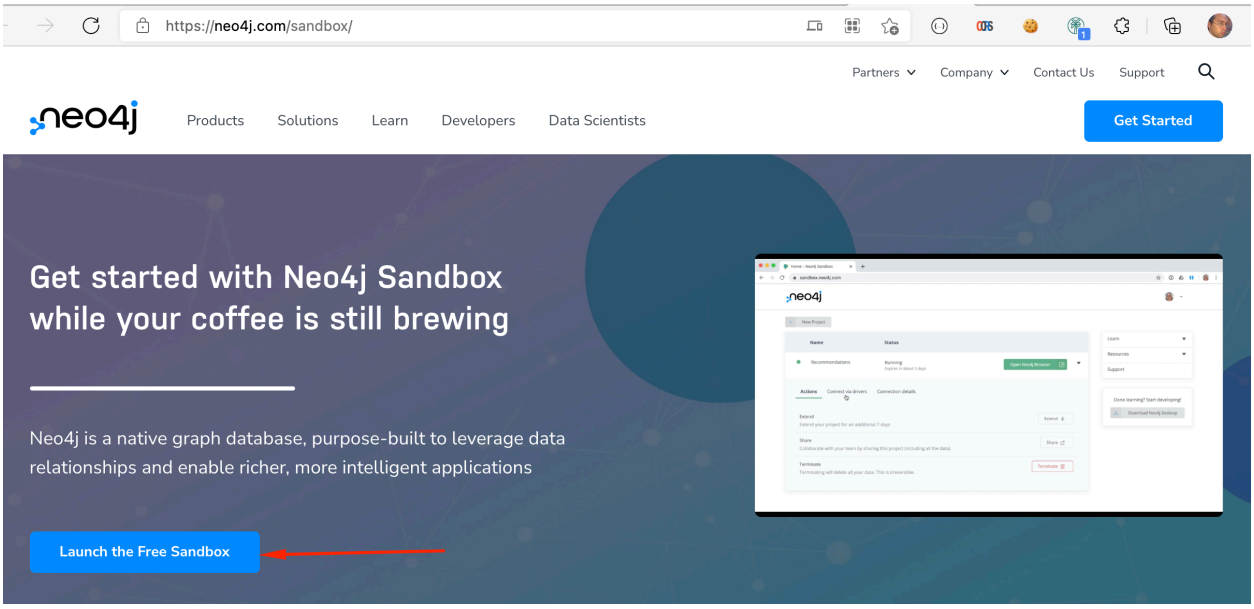


Tasks:

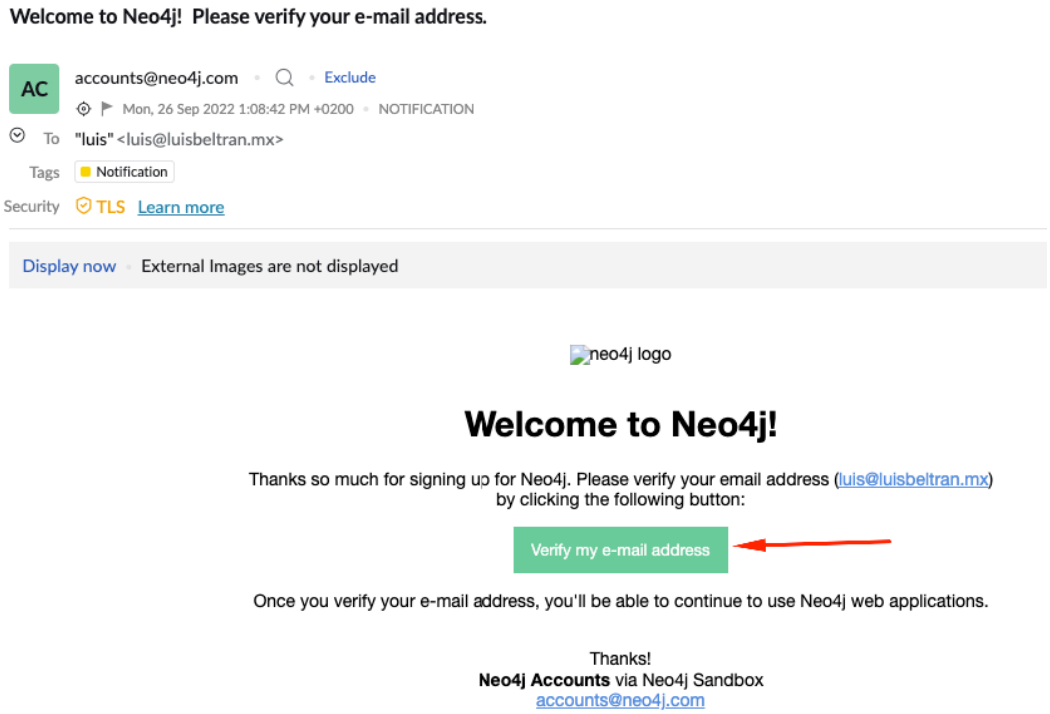
- Getting started with Neo4j Sandbox
- Creating a Project
- Our first Cypher query
- Understanding Nodes and Relationships
- Understanding Labels
- Understanding Properties
- Creating a Node
- Finding Nodes with Match and Where Clause
- Understanding the Merge Clause
- Creating a Relationship
- Relationship Types
- Advance Cypher queries

Task 1. Getting started with Neo4j Sandbox

Enter the following URL in a browser: **https://neo4j.com/sandbox/** . Click on **Launch the Free Sandbox**

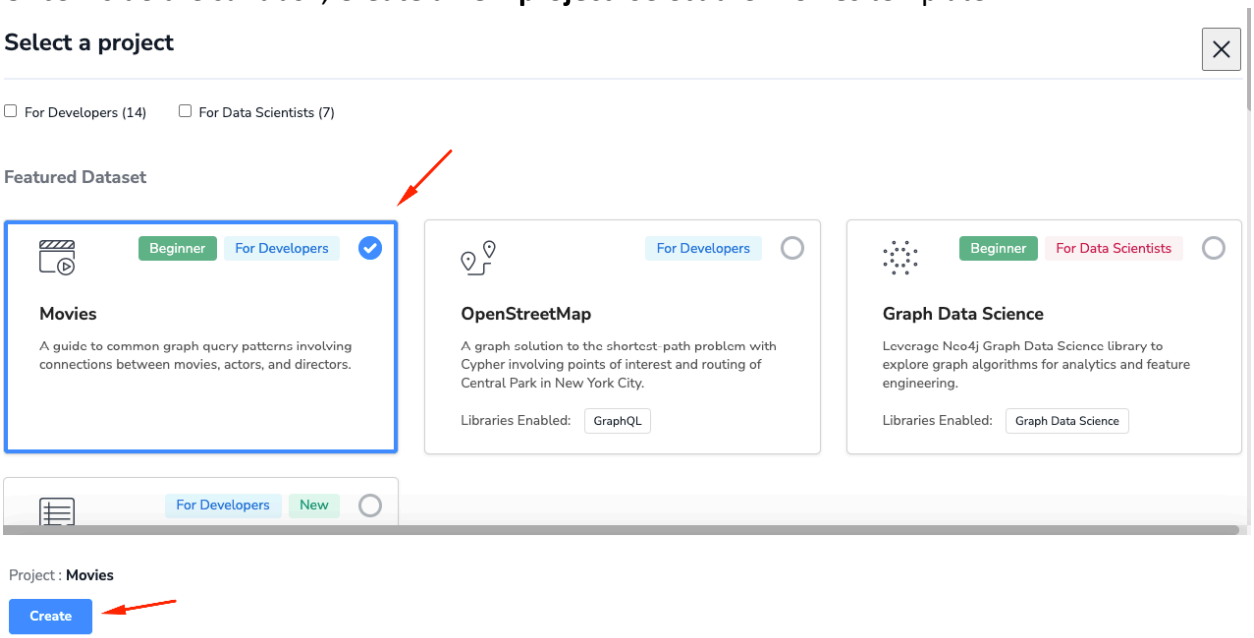


Sign up for a free account, follow the steps (it will ask you for basic information). After you finish, you'll receive a request to verify your account. Click on the link and you'll be all set.

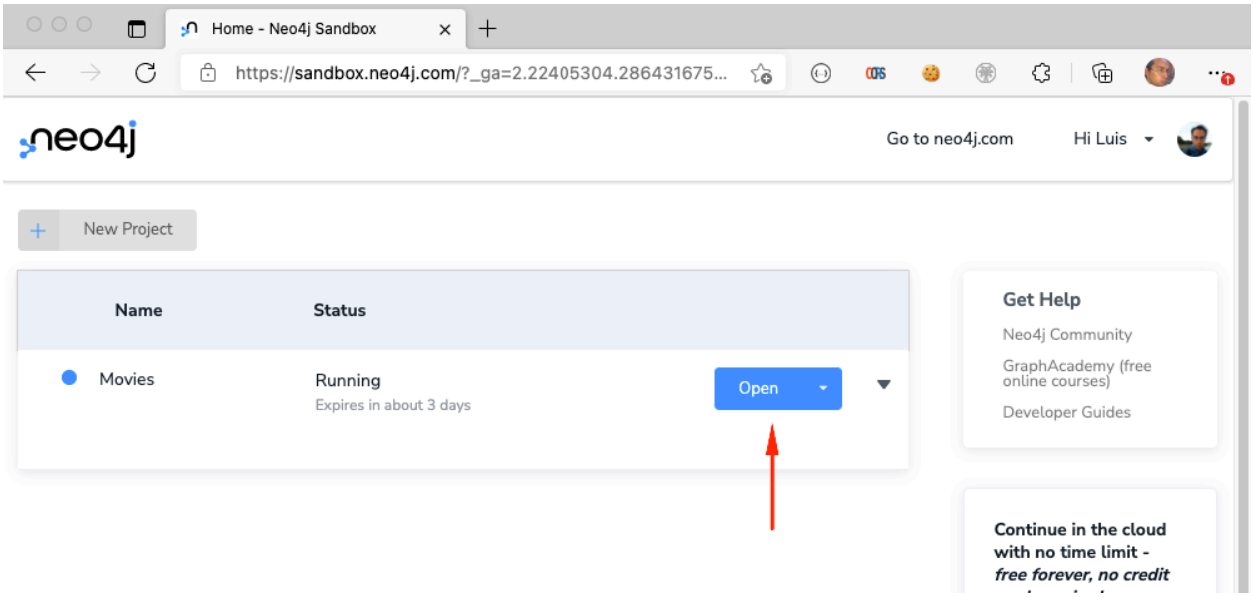


Task 2. Creating a Project

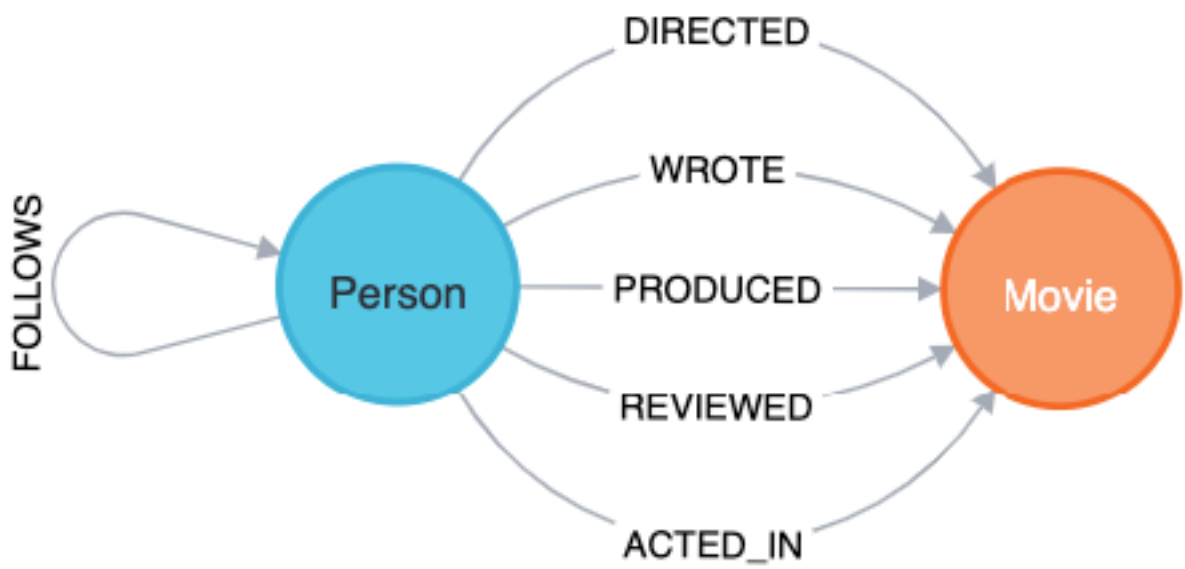
Once inside the sandbox, **Create a new project**. Select the **Movies** template.



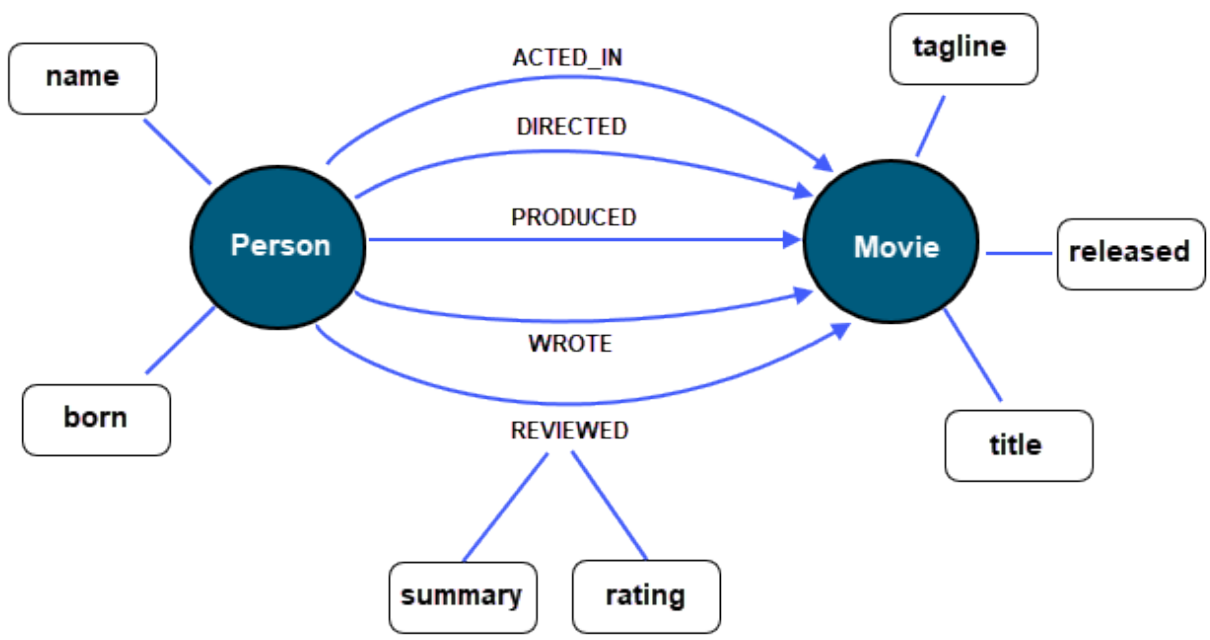
After the project has been created, **Open** it (it will take a couple of minutes).



The database, called **Movie Graph**, contains the actors and directors of some movies. In particular, there are 2 types of **nodes**: **Person** and **Movie**. These nodes are connected by a series of **relationships** that represent for example if an actor has **acted in** a movie or if the person has been the **director**. There are also social relationships such as whether a person is a **follower of** another on social networks. This is the structure of the Movies database



"Fields" of each node:



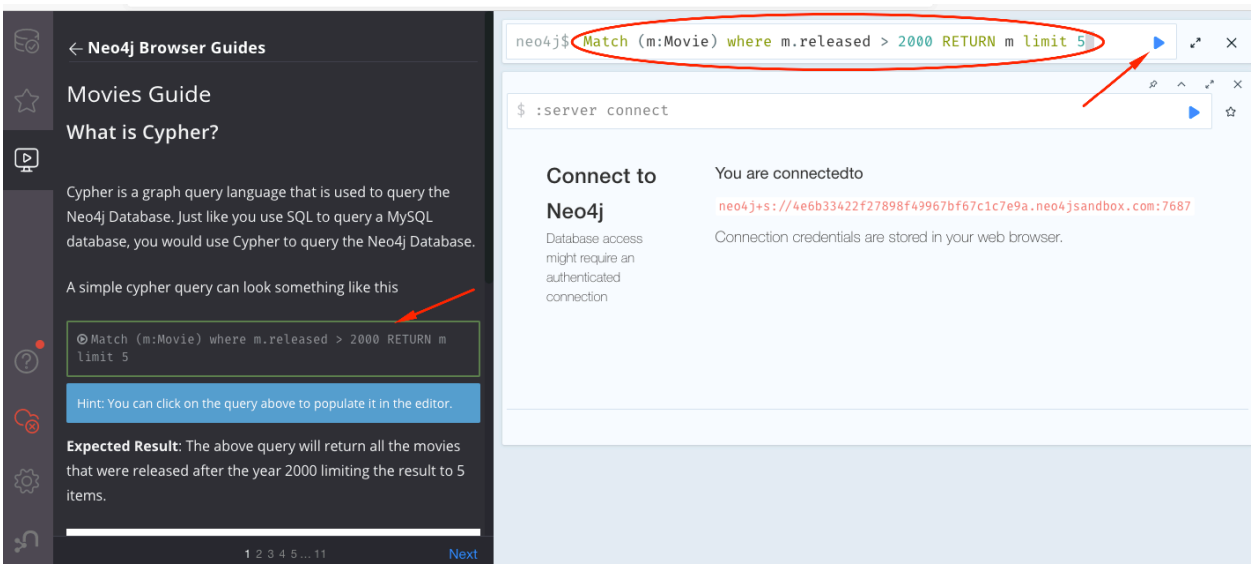
Task 3. Our first Cypher query

What is Cypher?

Cypher is a graph query language that is used to query the Neo4j Database. Just like you use SQL to query a MySQL database, you would use Cypher to query the Neo4j Database. It's a highly optimized language to find the nodes of interest and navigate the relationships between them. When writing a query in Cypher, one must remember that **there are no tables on which join operations must be performed**, only nodes and relationships. The idea must therefore be to identify the nodes of interest and from these navigate the available relations.

Let's try our first query. Click on the query that appears on the left on the tutorial. It looks like a typical SQL query.

```
Match (m:Movie) where m.released > 2000 RETURN m limit 5
```

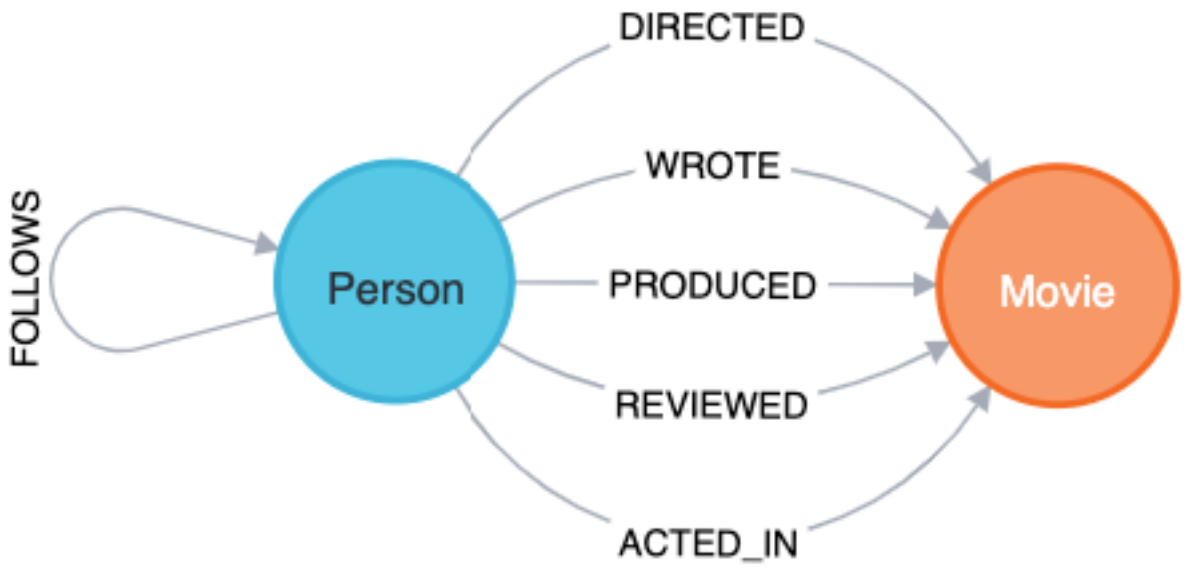


Result:

Exercise #1: Write a simple query to retrieve 3 members of the Person node which were born before 1980

Task 4. Understanding Nodes and Relationships
Nodes and Relationships are the basic building blocks of a graph database.

Nodes
Nodes represent entities. A node in graph database is similar to a row in a relational database. In the picture below we can see 2 kinds of nodes - Person and Movie.



Relationship
Two nodes can be connected with a relationship. In the above image ACTED_IN, REVIEWED, PRODUCED, WROTE and DIRECTED are all relationships connecting the corresponding types of nodes.

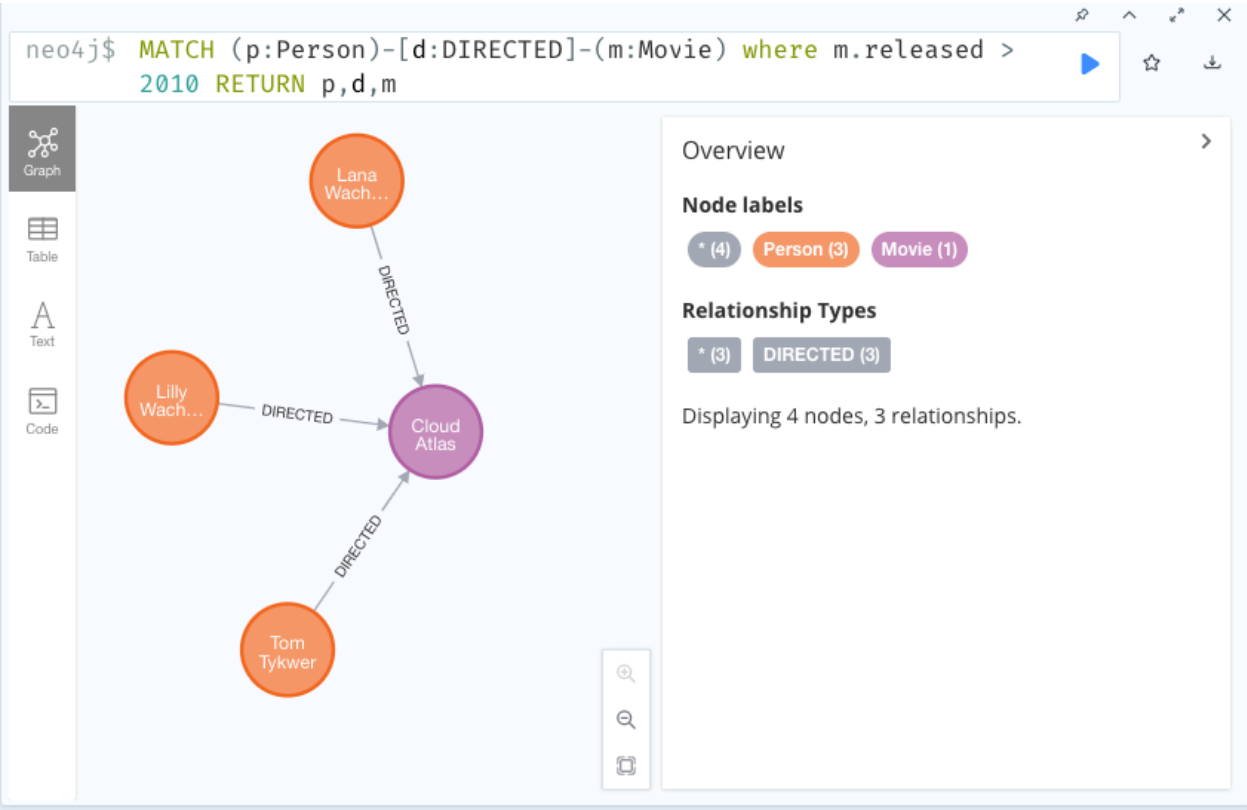
Two nodes can be connected with more than one relationships.

Cypher queries for nodes and relationships:

- A node is enclosed between a parenthesis — like (p:Person) where p is a variable and Person is the type of node it is referring to.
- Relationships are enclosed in square brackets - like [w:WORKS_FOR] where w is a variable and WORKS_FOR is the type of relationship it is referring to.

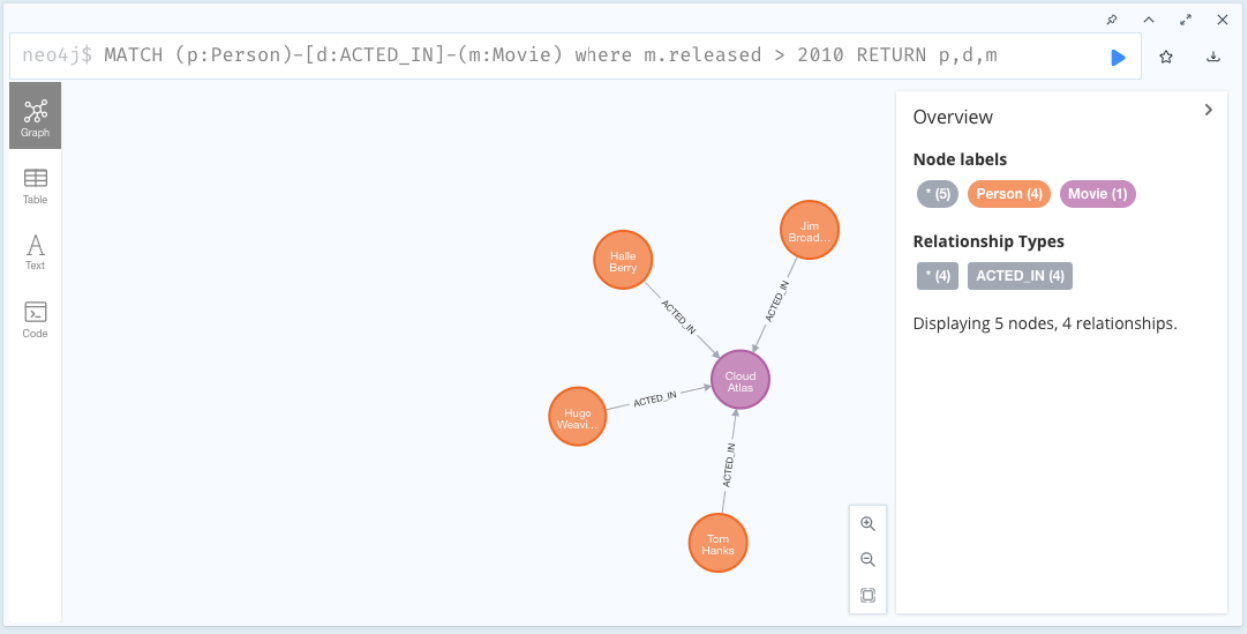
```
MATCH (p:Person)-[d:DIRECTED]-(m:Movie) where m.released > 2010
RETURN p,d,m
```

The above query will return all Person nodes who directed a movie that was released after 2010.



Query to get all the people who acted in a movie that was released after 2010.

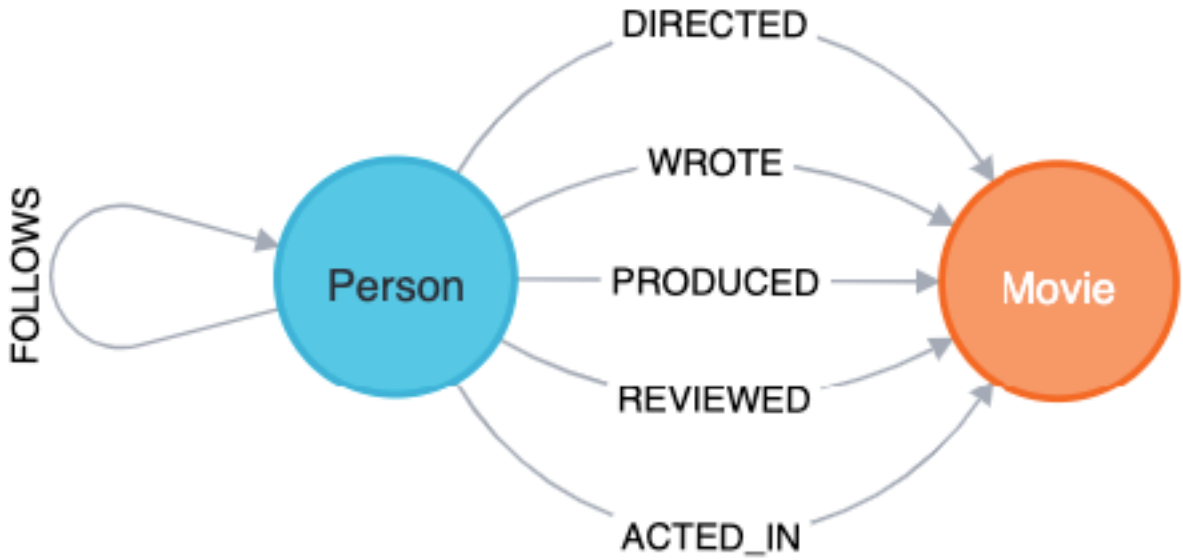
```
MATCH (p:Person)-[d:ACTED_IN]-(m:Movie) where m.released > 2010
RETURN p,d,m
```



Exercise #2: Write a query to get all the people who produced a movie that was released before 2005

Task 5. Understanding Labels

Labels is a name or identifer of a Node or a Relationship. In the image below Movie and Person are Node labels and ACTED_IN, REVIEWED, etc are Relationship labels.



In writing a cypher query, Labels are prefixed with a colon - like :Person or :ACTED_IN. You can assign the node label to a variable by prefixing the syntax with the variable name. Like (p:Person) means p variable denoted Person labeled nodes.

Labels are used when you want to perform operations only on a specific types of Nodes. Like

```
MATCH (p:Person) RETURN p limit 20
```

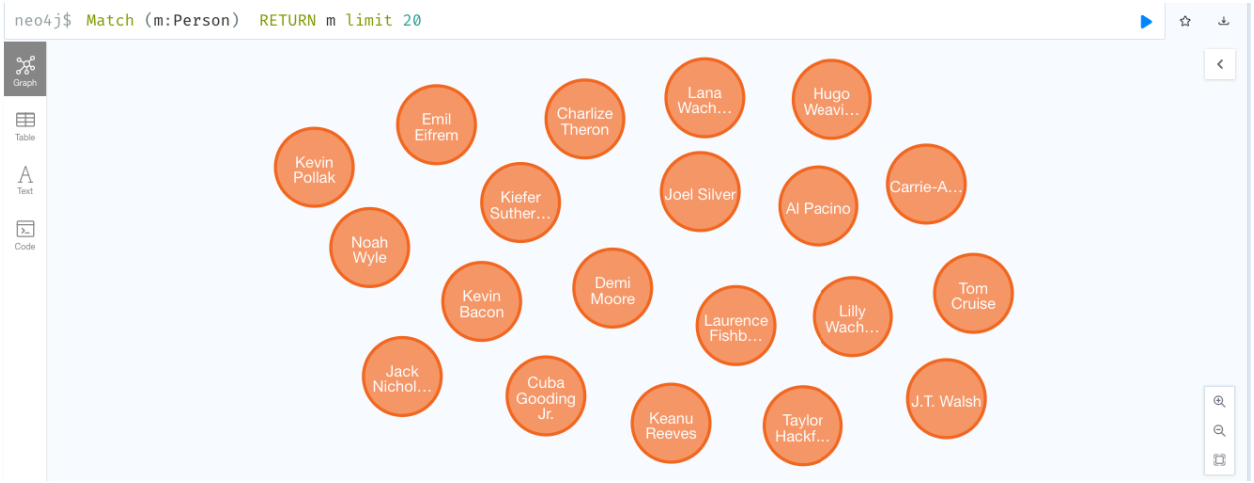
will return only Person Nodes (limiting to 20 items) while

```
MATCH (n) RETURN n limit 20
```

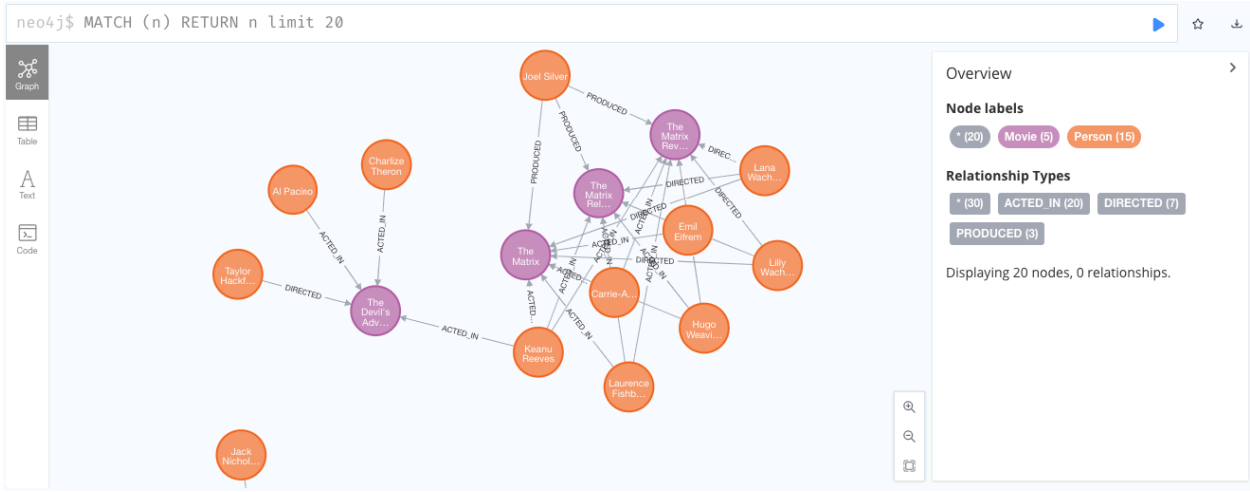
will return all kinds of nodes (limiting to 20 items).

Let's see the results:

MATCH (p:Person) RETURN p limit 20



MATCH (n) RETURN n limit 20



Exercise #3: Write a query that returns 10 Movies nodes

Task 6. Understanding Properties

Properties are name-value pairs that are used to add attributes to nodes and relationships.

To return specific properties of a node you can write -

```
MATCH (m:Movie) return m.title, m.released
```

This will return Movie nodes but with only the title and released properties.

```
neo4j$ MATCH (m:Movie) return m.title, m.released
```

	m.title	m.released
1	"The Matrix"	1999
2	"The Matrix Reloaded"	2003
3	"The Matrix Revolutions"	2003
4	"The Devil's Advocate"	1997
5	"A Few Good Men"	1992
6	"Top Gun"	1986
7		

Query to get name and born properties of the Person node.

```
MATCH (p:Person) return p.name, p.born
```

```
neo4j$ MATCH (p:Person) return p.name, p.born
```

	p.name	p.born
1	"Keanu Reeves"	1964
2	"Carrie-Anne Moss"	1967
3	"Laurence Fishburne"	1961
4	"Hugo Weaving"	1960
5	"Lilly Wachowski"	1967
6	"Lana Wachowski"	1965

Exercise #4: Write a query to get the name of actors who acted in movies that were released after 2005. Include the title of the movie

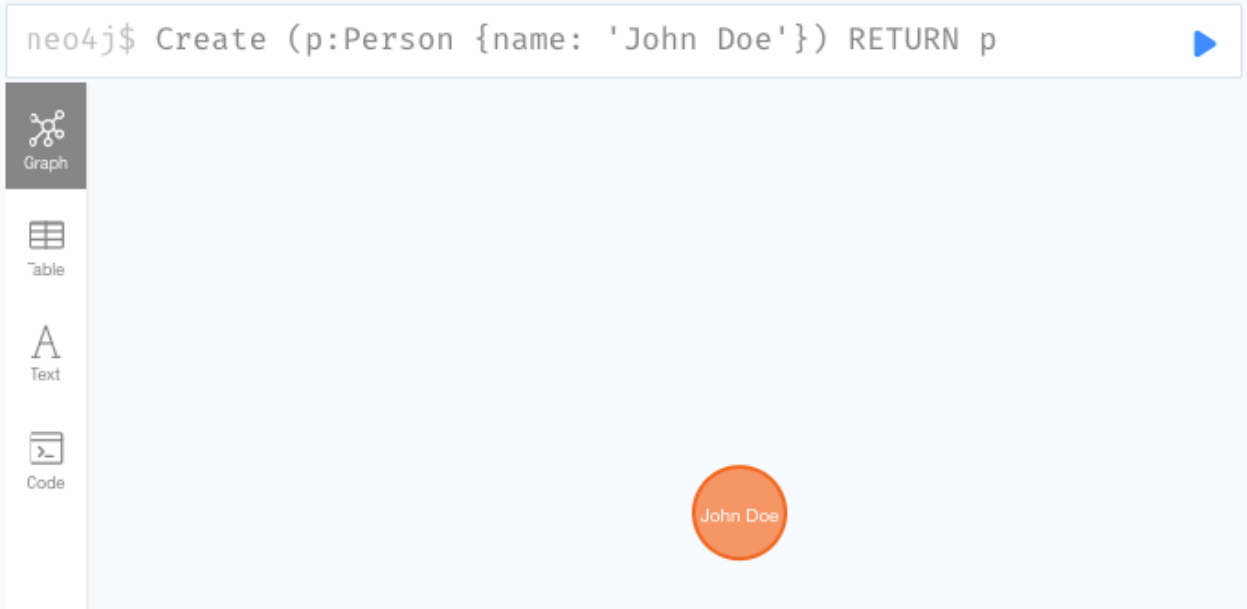
p.name	m.title
"Zach Grenier"	"RescueDawn"
"Steve Zahn"	"RescueDawn"
"Christian Bale"	"RescueDawn"
"Marshall Bell"	"RescueDawn"
"Tom Hanks"	"The Da Vinci Code"
"Ian McKellen"	"The Da Vinci Code"

Task 7. Creating a Node

Create clause can be used to create a new node or a relationship.

```
CREATE (p:Person {name: 'John Doe'}) RETURN p
```

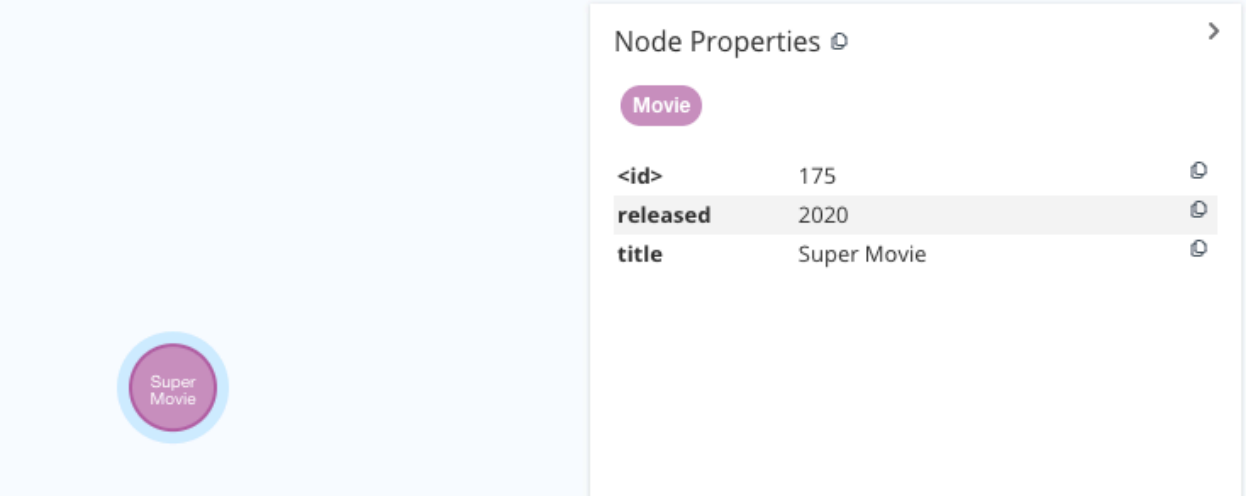
The above statement will create a new Person node with property name having value John Doe.



Create a new Person node with a property name having the value of your name.

```
CREATE (p:Person {name: '<Your Name>'}) RETURN p
```

Exercise #5: Create a new Movie node with properties title and released



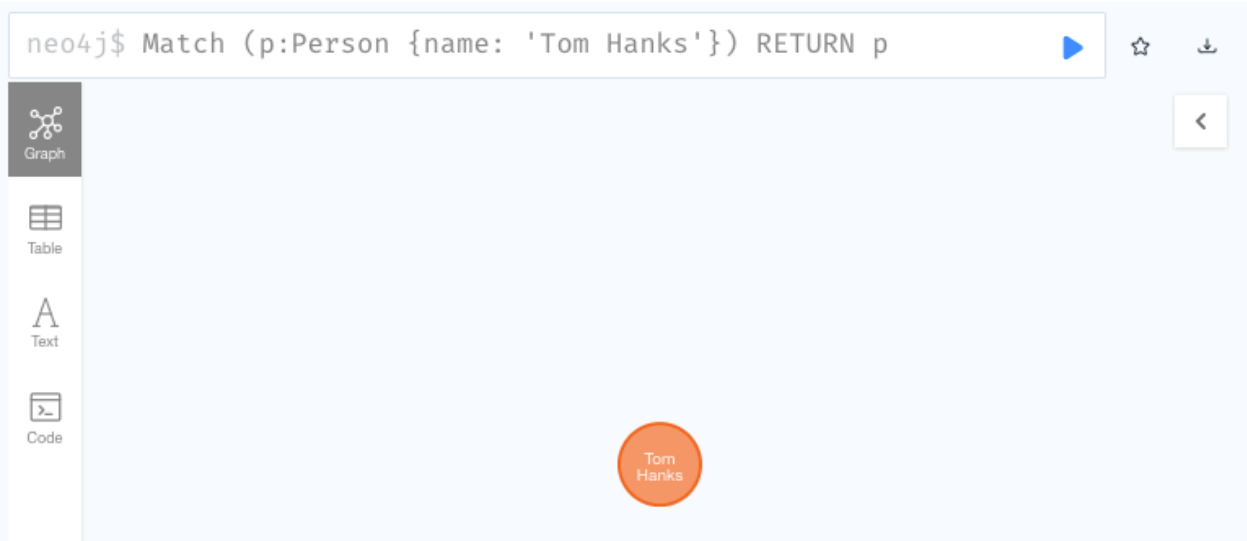
Task 8. Finding Nodes with Match and Where Clause

Match clause is used to find nodes that match a particular pattern. This is the primary way of getting data from a Neo4j database.

In most cases, a Match is used along with certain conditions to narrow down the result.

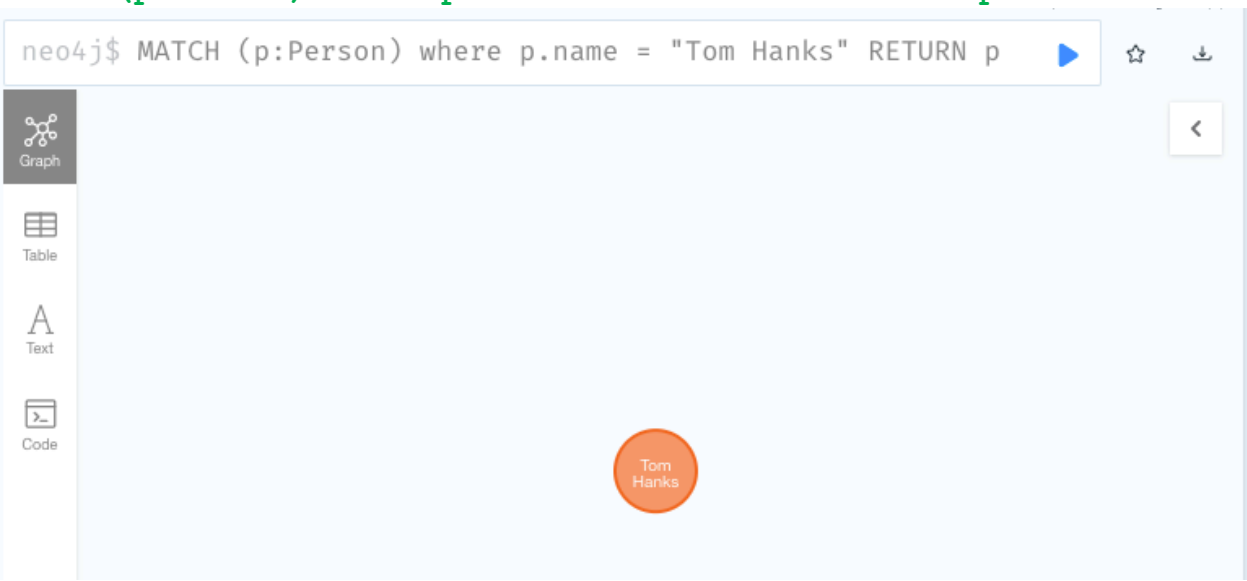
```
MATCH (p:Person {name: 'Tom Hanks'}) RETURN p
```

This is one way of doing it. Although you can only do basic string match based filtering this way (without using WHERE clause).



Another way would be to use a WHERE clause which allows for more complex filtering including >, <, Starts With, Ends With, etc

```
MATCH (p:Person) where p.name = "Tom Hanks" RETURN p
```



Both of the above queries will return the same results.

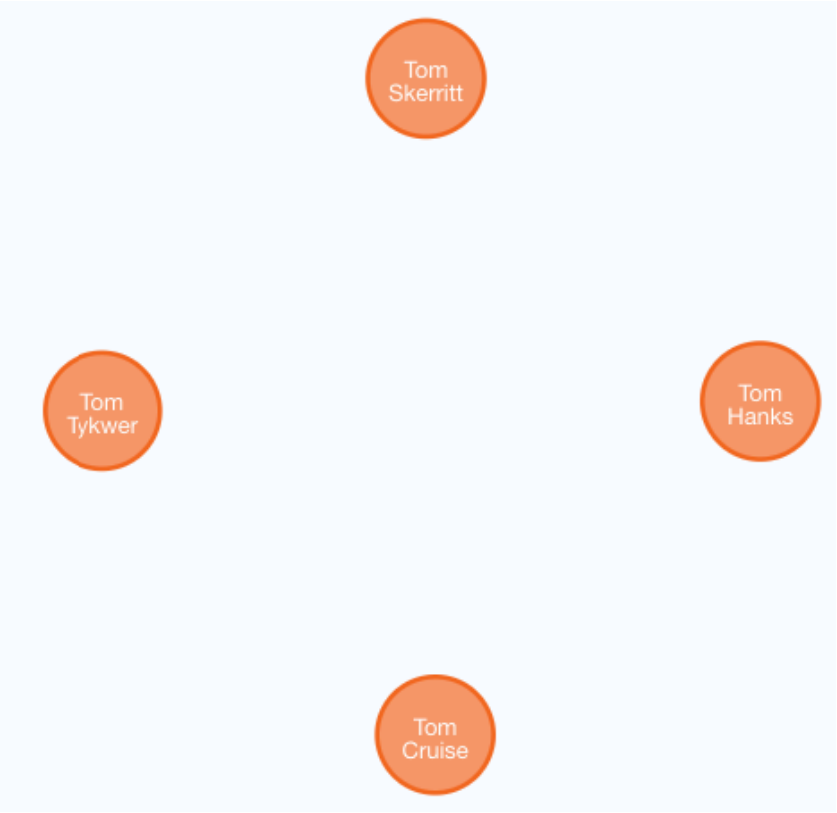
Find the movie with title "Cloud Atlas"

```
MATCH (m:Movie {title: "Cloud Atlas"}) return m
```

Get all the movies that were released between 2010 and 2015.

```
MATCH (m:Movie) where m.released > 2010 and m.released < 2015
RETURN m
```

Exercise #6: Write a query that retrieves all Persons named Tom



Task 9. Understanding the Merge Clause

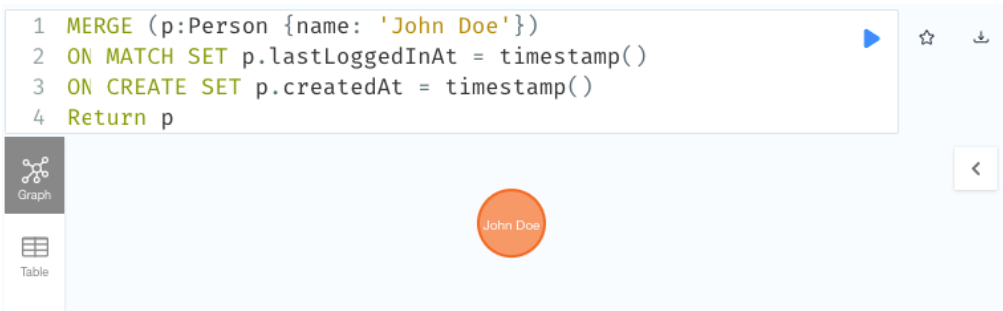
The Merge clause is used to either

- Match the existing nodes and bind them or
- Create new node(s) and bind them

It is a combination of Match and Create and additionally allows to specify additional actions if the data was matched or created.

```
MERGE (p:Person {name: 'John Doe'})
ON MATCH SET p.lastLoggedInAt = timestamp()
ON CREATE SET p.createdAt = timestamp()
RETURN p
```

The above statement will create the Person node if it does not exist. If the node already exists, then it will set the property lastLoggedInAt to the current timestamp. If node did not exist and was newly created instead, then it will set the createdAt property to the current timestamp.

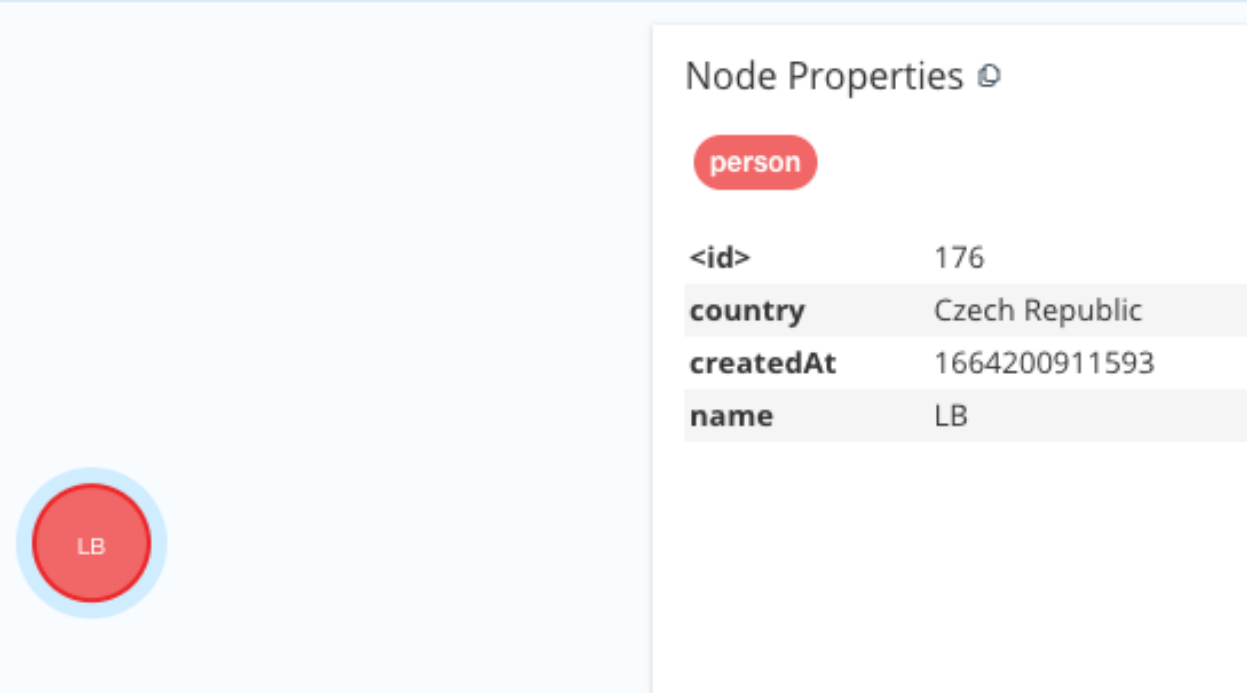


See its properties (click on Table)

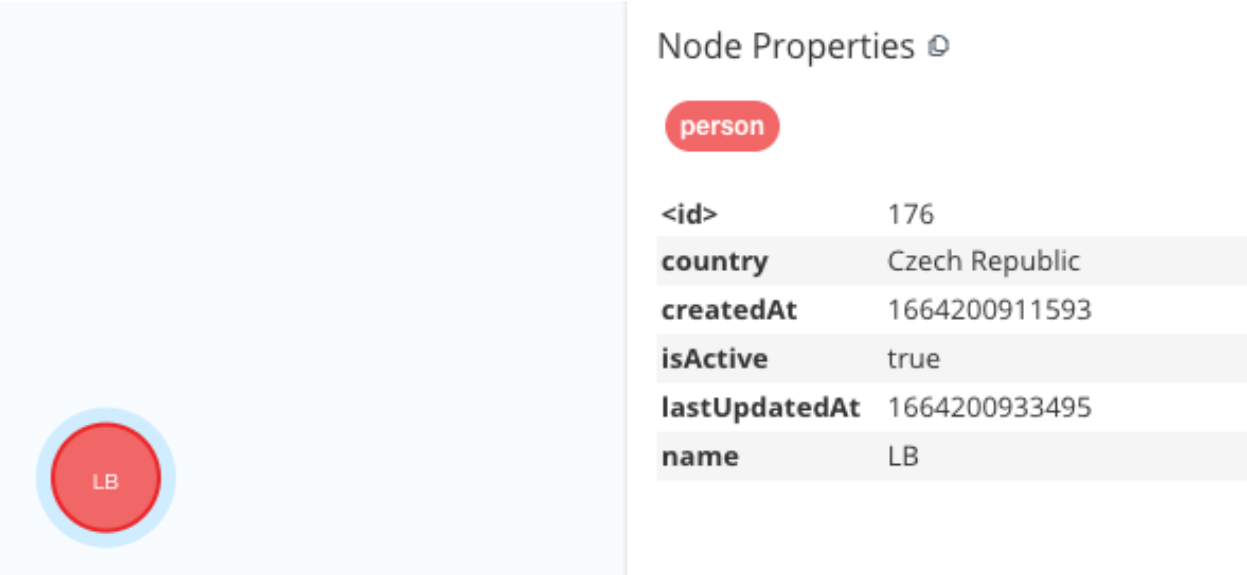
Query using Merge to create a movie node with title "Greyhound". If the node does not exist, then set its released property to 2020 and lastUpdatedAt property to the current time stamp. If the node already exists, then only set lastUpdatedAt to the current time stamp. Return the movie node.

```
MERGE (m:movie {title: 'Greyhound'})
ON MATCH SET m.lastUpdatedAt = timestamp()
ON CREATE SET m.released = "2020", m.lastUpdatedAt = timestamp()
RETURN m
```

Exercise #7: Write a query that creates a Person node with any name. If the node does not exist, then set its country property to 'Czech Republic' and createdAt property to the current time stamp. If the node already exists, then set isActive to true and lastUpdatedAt to the current time stamp. Return the Person node.



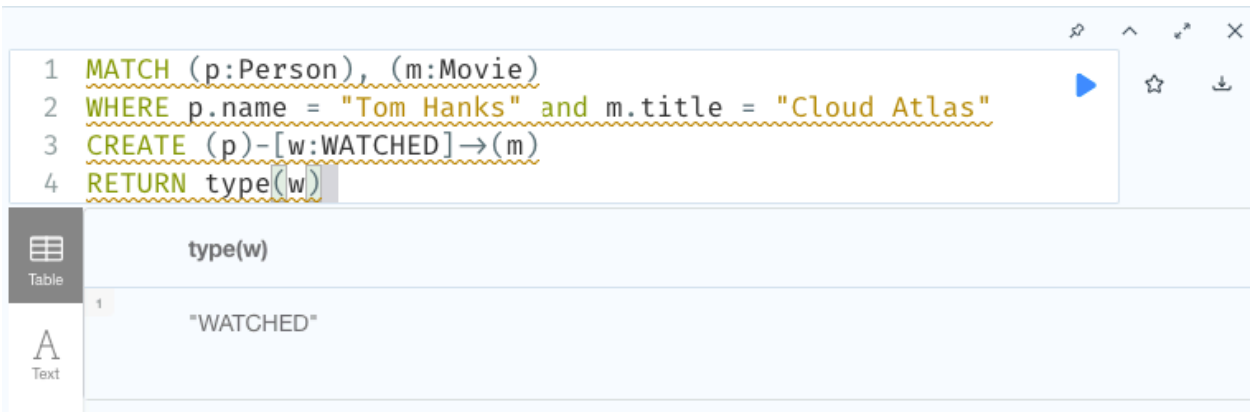
Ejecutando la consulta por segunda occasion…



Task 10. Creating a Relationship
A Relationship connects 2 nodes.

```
MATCH (p:Person), (m:Movie)
WHERE p.name = "Tom Hanks" and m.title = "Cloud Atlas"
CREATE (p)-[w:WATCHED]->(m)
RETURN type(w)
```

The above statement will create a relationship :WATCHED between the existing Person and Movie nodes and return the type of relationship (i.e WATCHED).



Create a relationship :WATCHED between the node you created for yourself previously and the movie Cloud Atlas and then return the type of created relationship

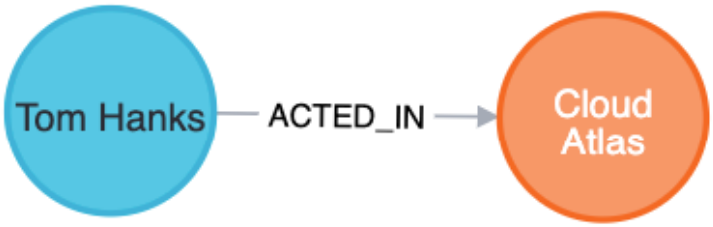
```
MATCH (p:Person), (m:Movie)
WHERE p.name = "<Your Name>" and m.title = "Cloud Atlas"
CREATE (p)-[w:WATCHED]->(m)
RETURN type(w)
```

Exercise #8: Write a query that creates a relationship :FAN OF between the nodes you created previously (Person in Task #7 and Movie from Exercise #5) and then return the type of created relationship

type(w)
"FAN_OF"

Task 11. Relationship Types

In Neo4j, there can be 2 kinds of relationships - incoming and outgoing.



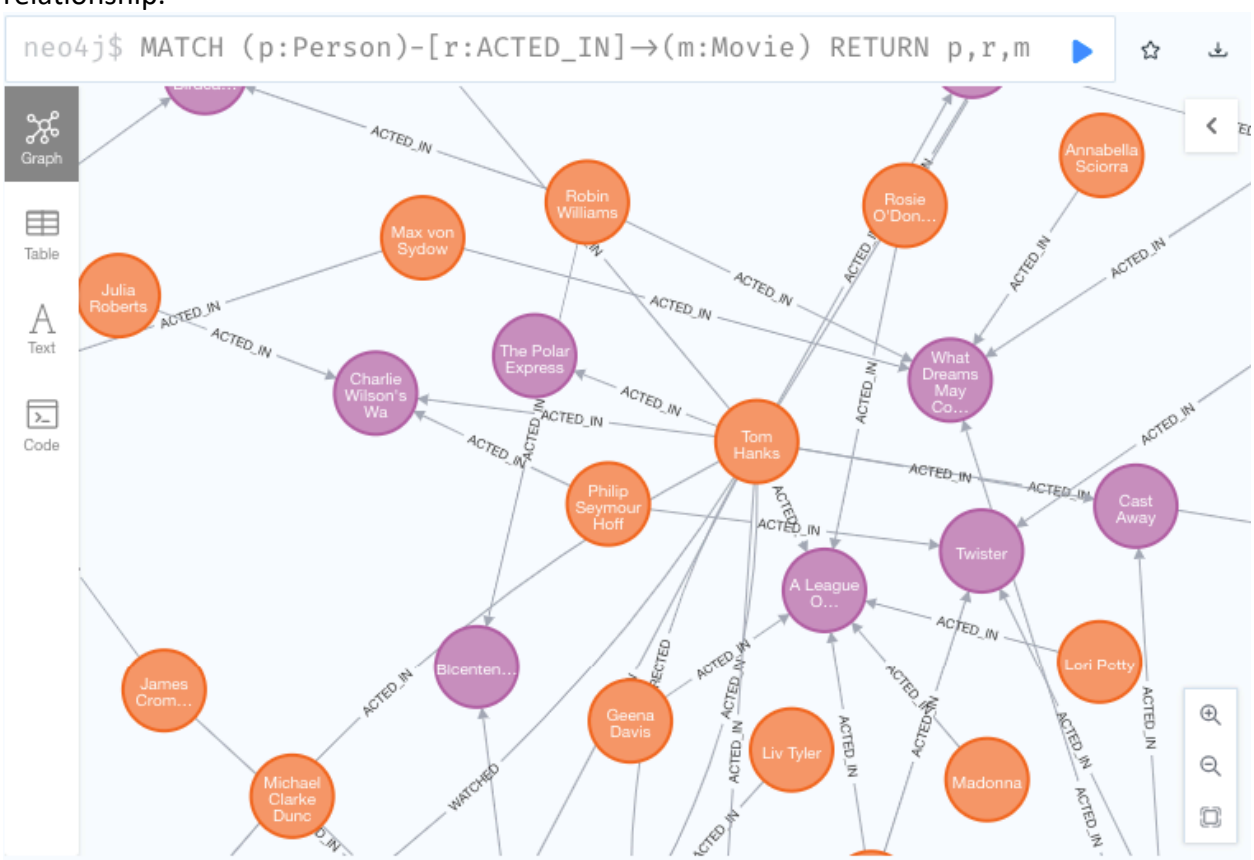
In the above picture, the Tom Hanks node is said to have an outgoing relationship while Cloud Atlas node is said to have an incoming relationship.

Relationships always have a direction. However, you only have to pay attention to the direction where it is useful.

To denote an outgoing or an incoming relationship in cypher, we use → or ←.

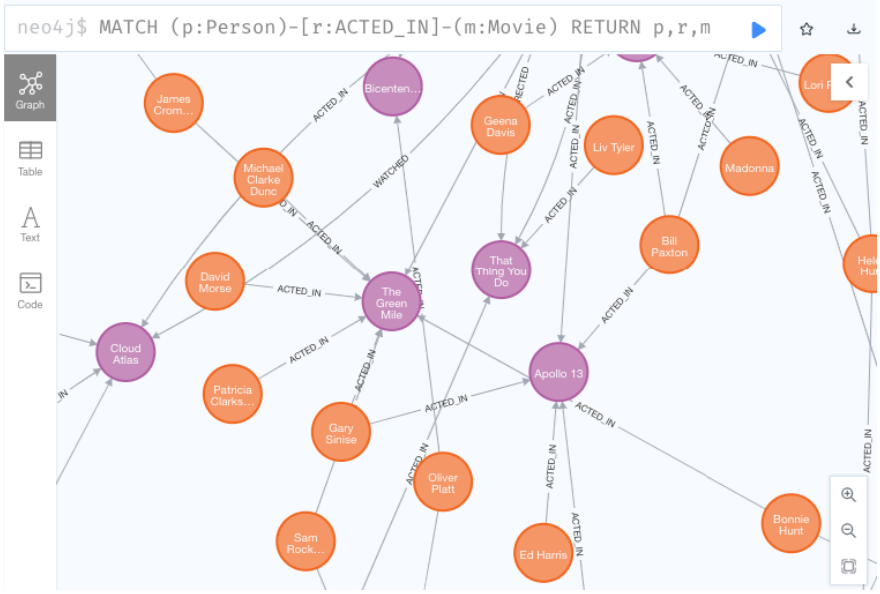
```
MATCH (p:Person)-[r:ACTED_IN]->(m:Movie) RETURN p,r,m
```

In the above query Person has an outgoing relationship and movie has an incoming relationship.



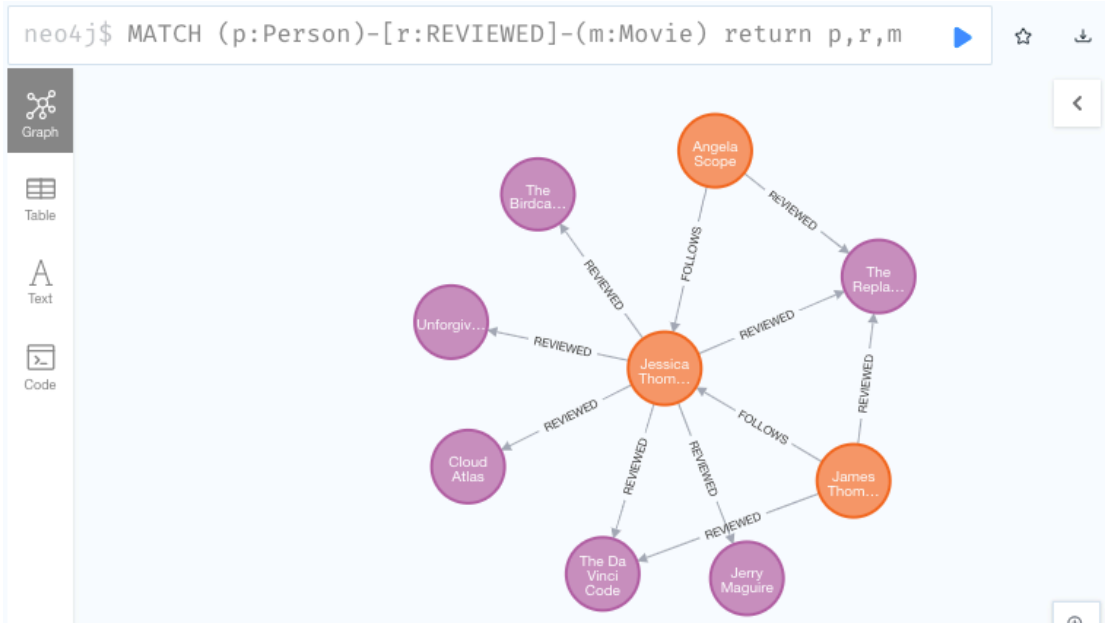
Although, in the case of the movies dataset, the direction of the relationship is not that important and even without denoting the direction in the query, it will return the same result. So the following query will return the same result as the above one.

```
MATCH (p:Person)-[r:ACTED_IN]-(m:Movie) RETURN p,r,m
```

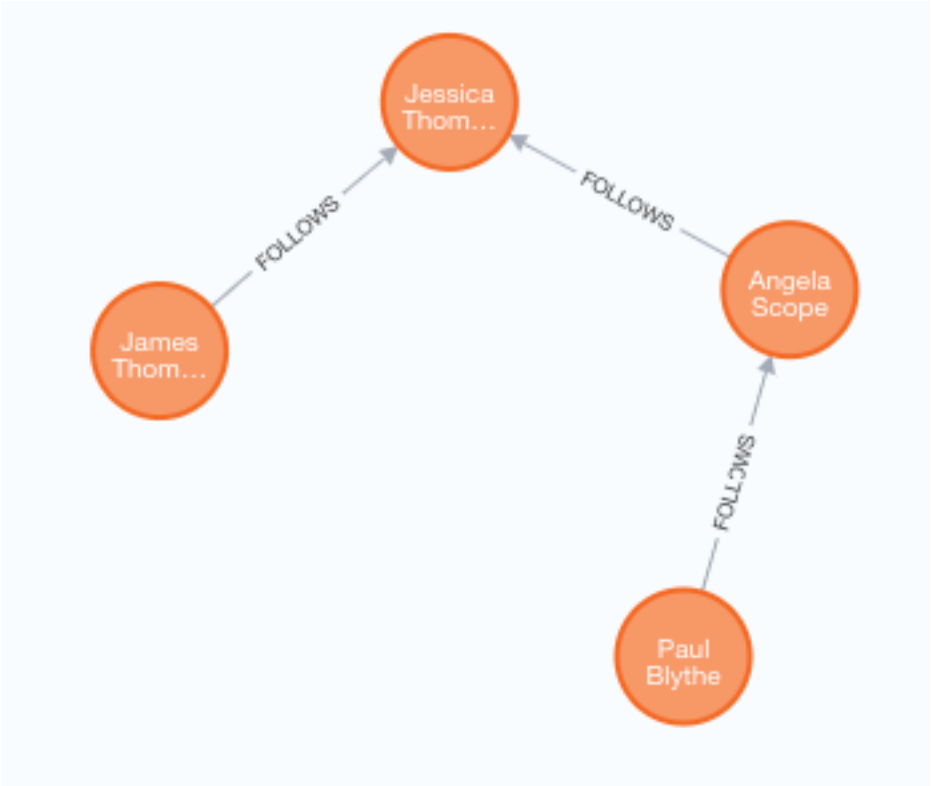


Query to find the nodes Person and Movie which are connected by REVIEWED relationship and is outgoing from the Person node and incoming to the Movie node.

```
MATCH (p:Person)-[r:REVIEWED]-(m:Movie) return p,r,m
```



Exercise #9: Write a query to find the nodes Person which are connected by FOLLOWS relationship



Task 12. Advance Cypher queries

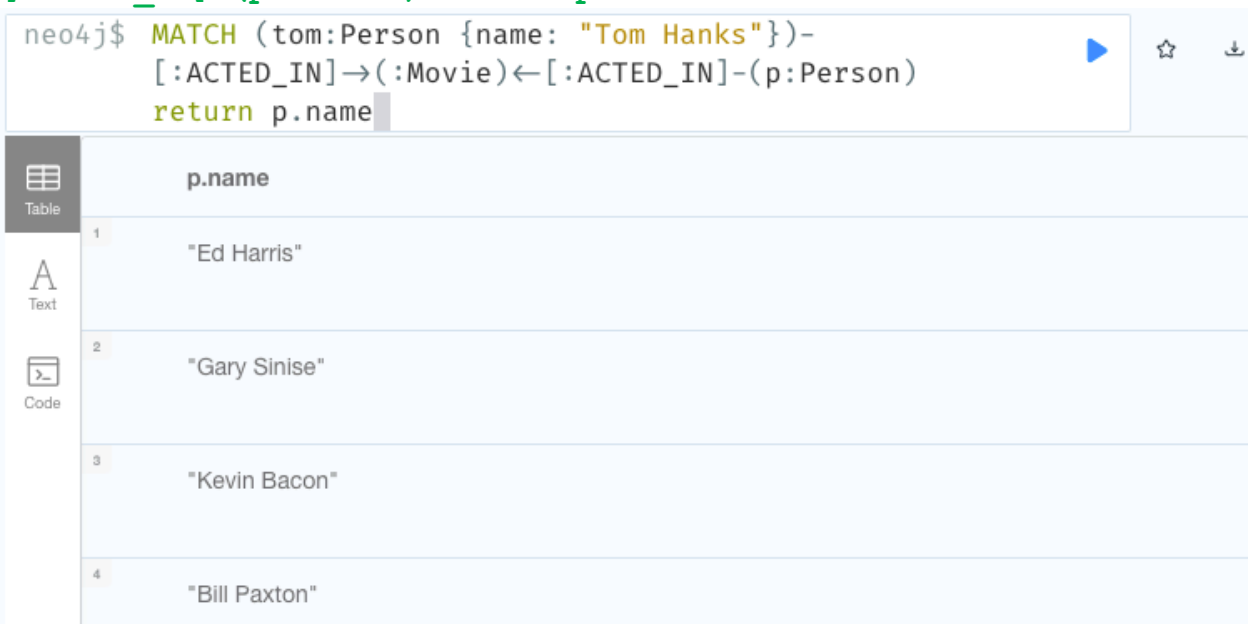
Let's look at some questions that you can answer with cypher queries.

Finding who directed Cloud Atlas movie

```
MATCH (m:Movie {title: 'Cloud Atlas'})<-[d:DIRECTED]-(p:Person)
return p.name
```

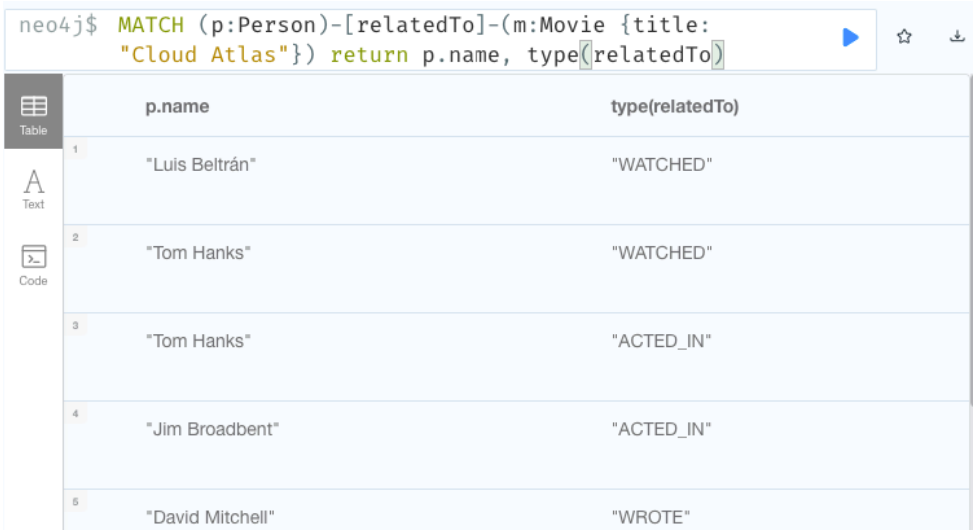
Finding all people who have co-acted with Tom Hanks in any movie

```
MATCH (tom:Person {name: "Tom Hanks"})-[:ACTED_IN]->(:Movie)<-
[:ACTED_IN]-(p:Person) return p.name
```



Finding all people related to the movie Cloud Atlas in any way

```
MATCH (p:Person)-[relatedTo]-(m:Movie {title: "Cloud Atlas"})
return p.name, type(relatedTo)
```



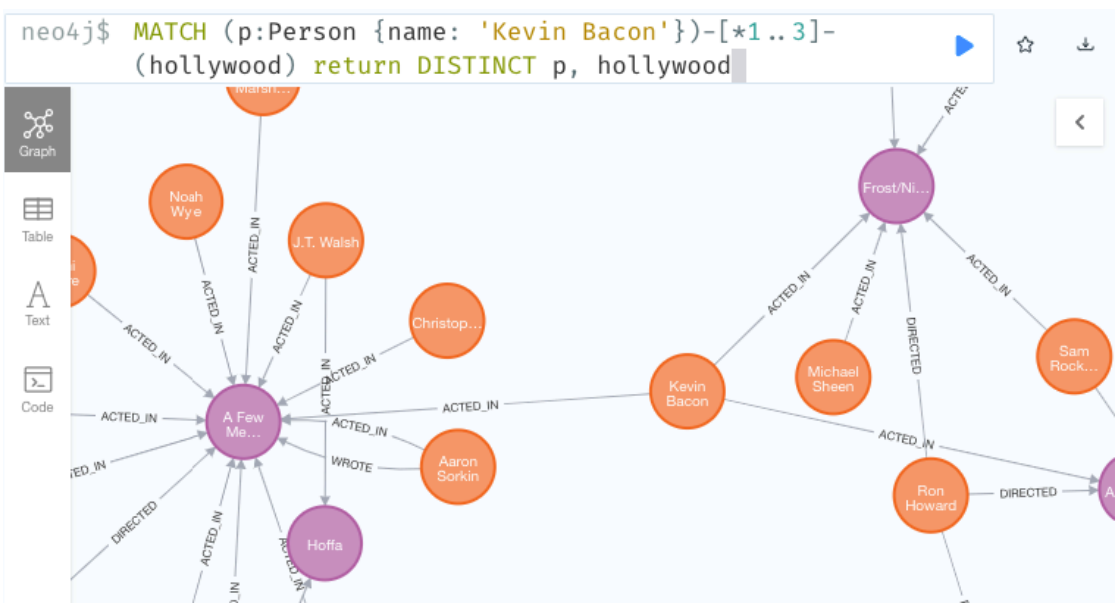
	p.name	type(relatedTo)
1	"Luis Beltrán"	"WATCHED"
2	"Tom Hanks"	"WATCHED"
3	"Tom Hanks"	"ACTED_IN"
4	"Jim Broadbent"	"ACTED_IN"
5	"David Mitchell"	"WROTE"

In the above query we only used the variable relatedTo which will try to find all the relationships between any Person node and the movie node "Cloud Atlas"

Finding Movies and Actors that are 3 hops away from Kevin Bacon.

```
MATCH (p:Person {name: 'Kevin Bacon'})-[*1..3]-(hollywood)
return DISTINCT p, hollywood
```

Note: in the above query, hollywood refers to any node in the database (in this case Person and Movie nodes)



Exercise #10: Write a query to find who reviewed the movie “Cloud Atlas”

p.name
"Jessica Thompson"

Write a second query that returns all movies directed by Tom Hanks and where he acted in

m.title
"That Thing You Do"

Write a final query that returns all actors who have a maximum of “Three Degrees of Tom Hanks”