# Cypher Fundamentals course notes and links-

Course Link: <https://graphacademy.neo4j.com/courses/cypher-fundamentals/>

Video Introduction: <https://youtu.be/Se_Zwiew90Q>

# Module 1: Reading Data from Neo4j

In this module you will learn how to write Cypher code to retrieve data from the graph.

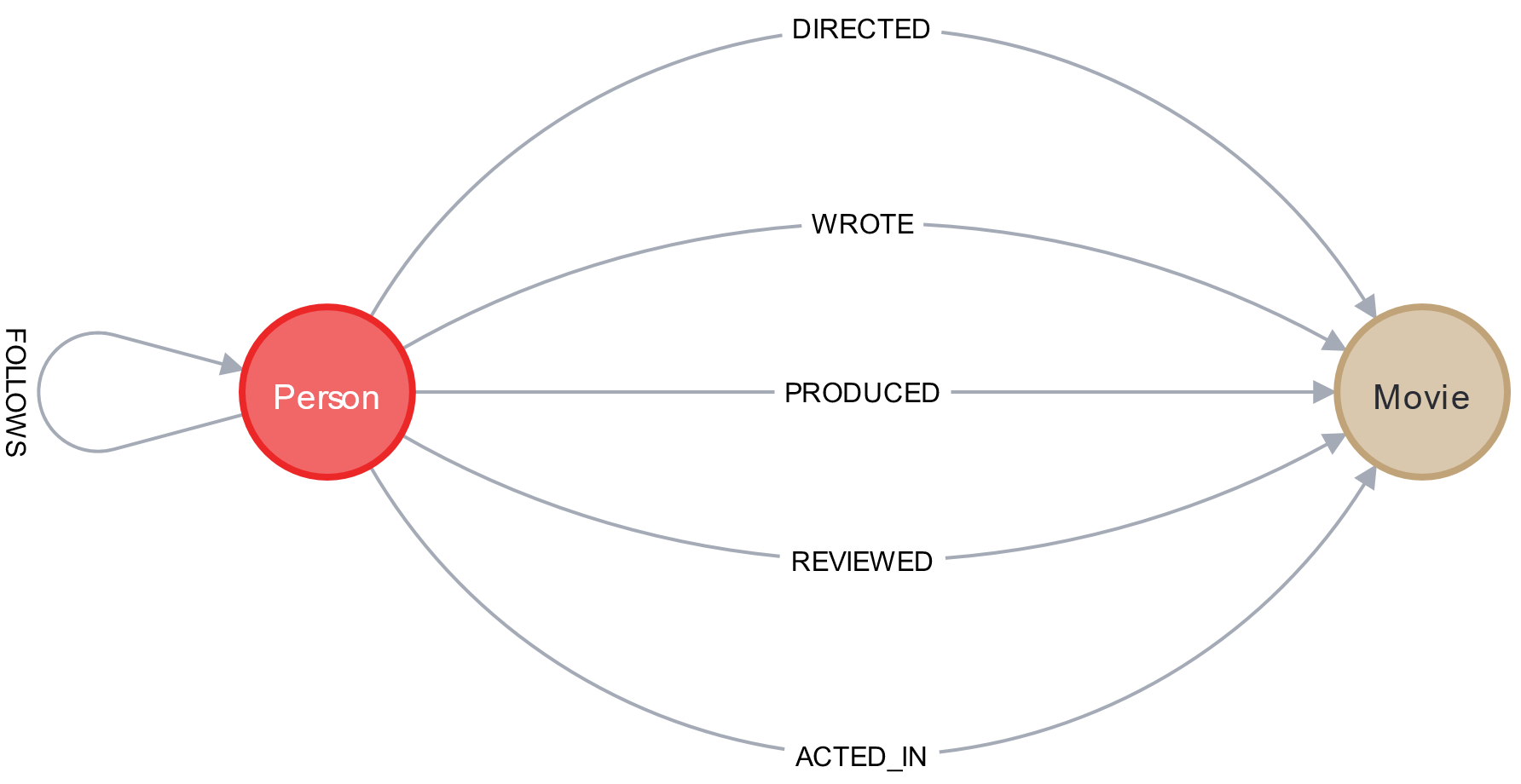
You will learn how to:

* Retrieve nodes from the graph.
  + Retrieve nodes with a particular label.
  + Filter the retrieval by a property value.
  + Return property values.
* Retrieve nodes and relationships from the graph using patterns in the graph.
* Filter queries

Using the Movies example dataset, you will create and execute Cypher code to find actors and movies in our graph.

## Domain model for this course

Here is the data model used in this course. The graph contains nodes with the labels Person and Movie. Person nodes have several types of relationships to Movie nodes. A Person node can have a FOLLOWS relationship to another Person node.



# Introduction to Cypher

Youtube link: <https://youtu.be/jEIE_b1MzAE>

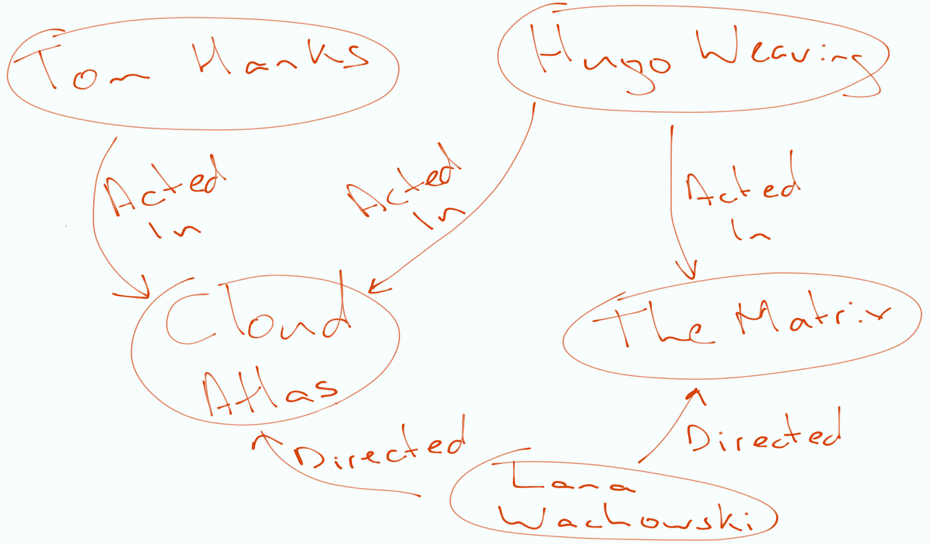
Transcript-

Sandbox neo4j lesson is there and contains screenshots

## What is Cypher?

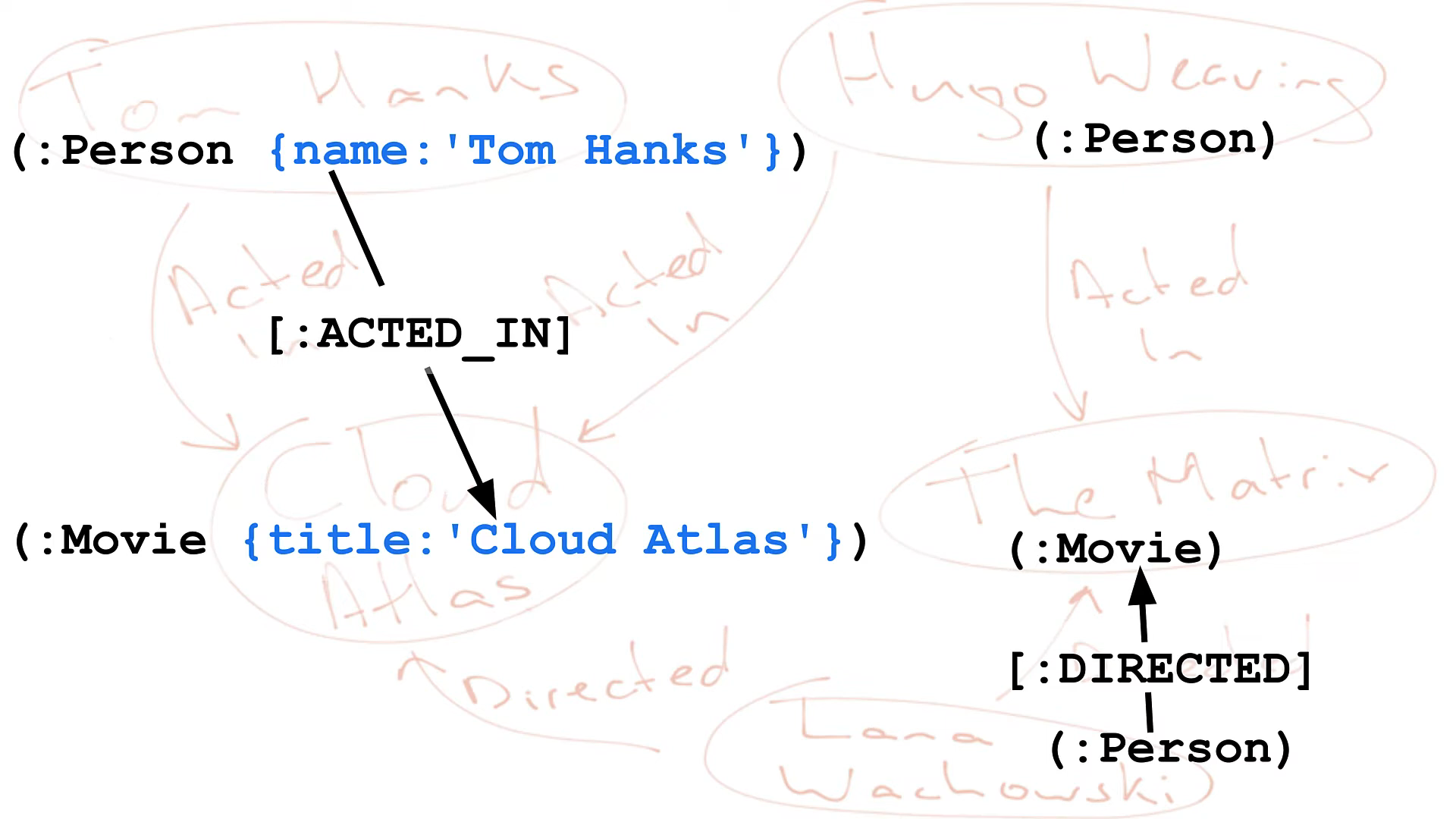
Cypher is a query language designed for graphs.

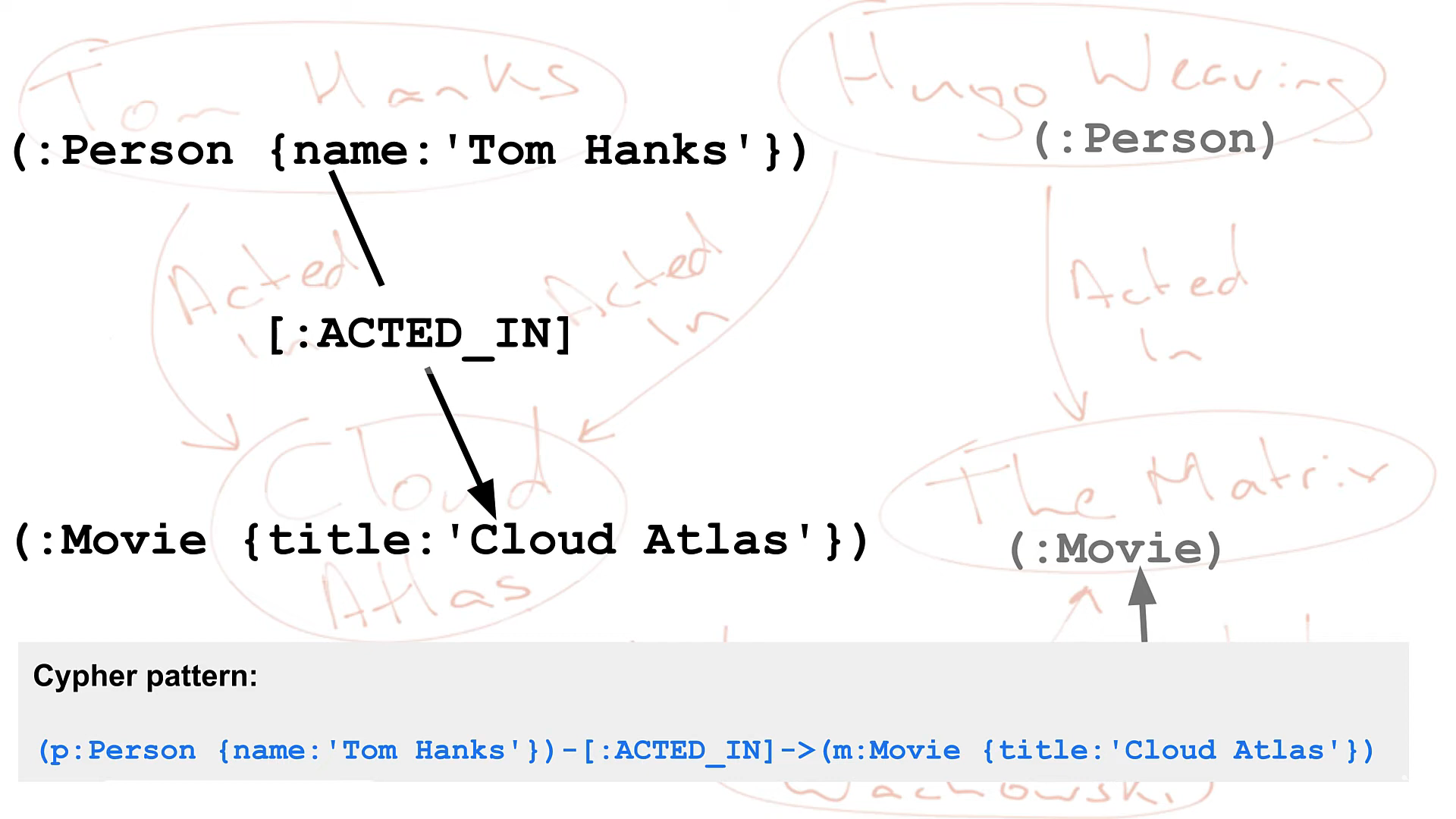
The whiteboard model of our domain entities is stored in the database as a graph. When we draw a graph on the whiteboard, we represent entities as circles connected together using arrows. In this example, the entities are people and movies. We have Person and Movie nodes in our graph.



Just as we would draw circles and arrows on a whiteboard, we write out the pattern in Cypher:

* Nodes are represented by parentheses ().
* We use a colon to signify the label(s), for example (:Person).
* Relationships between nodes are written with two dashes, for example (:Person)--(:Movie).
* The direction of a relationship is indicated using a greater than or less than symbol < or > , for example (:Person)-→(:Movie).
* The type of the relationship is written using the square brackets between the two dashes: [ and ], for example [:ACTED\_IN]
* Properties drawn in a speech bubble are specified in a JSON like syntax.
  + Properties in Neo4j are key/value pairs, for example {name: 'Tom Hanks'}.





For example, a Cypher pattern in the graph could be:

**Partial**

// example Cypher pattern

(m:Movie {title: 'Cloud Atlas'})<-[:ACTED\_IN]-(p:Person)

The two node types in this pattern are Movie and Person. The Person nodes have a directed ACTED\_IN relationship to Movie nodes. The specific Movie node in this pattern is filtered by the title property with a value of Cloud Atlas. So this pattern represents all people in the graph who acted in the movie, Cloud Atlas.

### How Cypher works

Cypher works by matching patterns in the data. We retrieve data from the graph using the MATCH keyword. You can think of the MATCH clause as similar to the FROM clause in an SQL statement.

For example, if we want to find a Person in the graph, we would MATCH a pattern of a single node with a label of :Person - prefixed with a colon :.

**Partial**

MATCH (:Person)

// incomplete MATCH clause because we need to return something

Suppose we want to retrieve all Person nodes from the graph. We can assign a variable by placing a value before the colon. Let’s use the variable p. Now that p represents all Person nodes retrieved from the graph, we can return them using the RETURN clause.

Run this Cypher code:

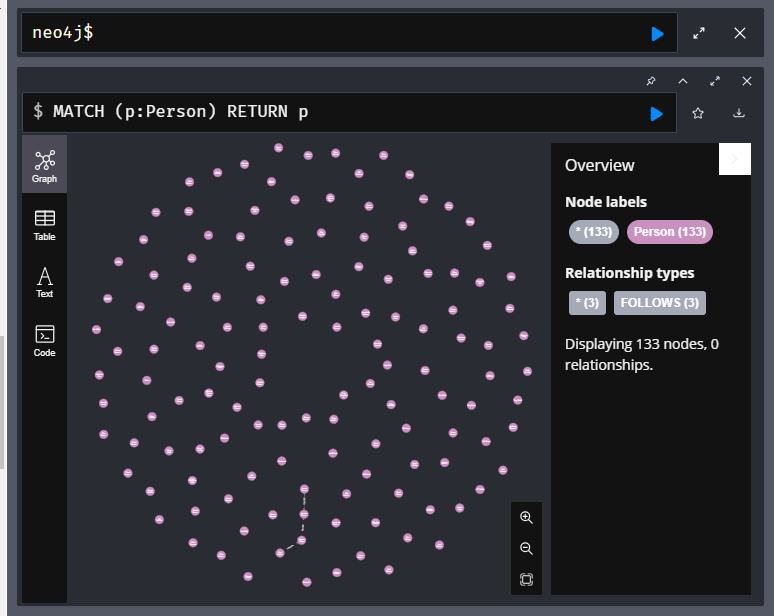
Click the **Run in Sandbox** button to the top right of the code sample to open the Sandbox to the right and run the query.

**cypher**

Copy to ClipboardRun in Sandbox

MATCH (p:Person)

RETURN p



This query returns all nodes in the graph with the Person label. You can view the results returned using the graph view or the table view. When you select the table view, you can also see the properties for the nodes returned.

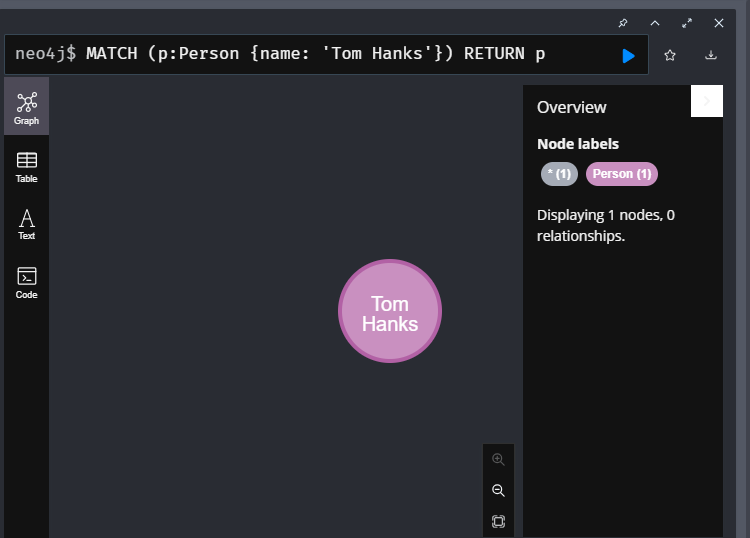
Now, say we want to find the node which represents the Person who’s name is Tom Hanks. Our Person nodes all have a name property. We can use the braces {..} to specify the key/value pair of name and Tom Hanks as the filter. As Tom Hanks is a string, we will need to place it inside single or double quotes.

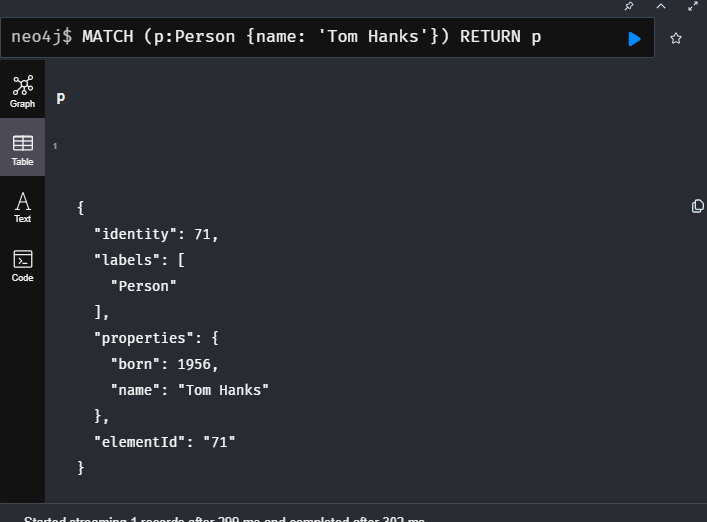
**cypher**

Copy to ClipboardRun in Sandbox

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

RETURN p





This query returns a single node that represents Tom Hanks. In the graph view of Neo4j Browser, the node is visualized as a bubble. You can also view the results returned in table view where you can view the properties of the node.

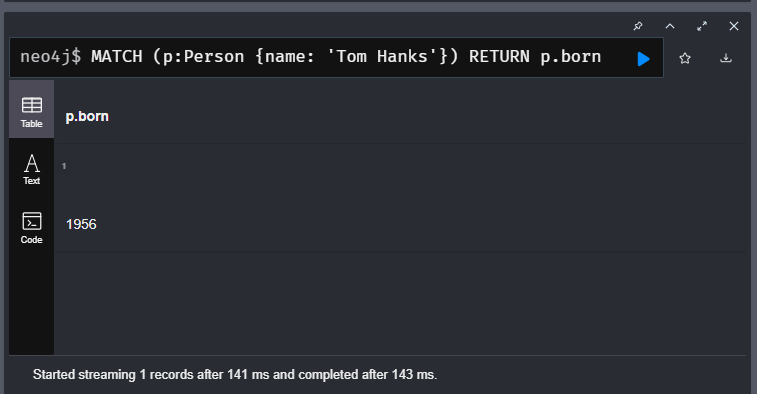
In our Cypher statement, we can access properties using a dot notation. For example, to return the name property value using its property key p.name.

**cypher**

Copy to ClipboardRun in Sandbox

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

RETURN p.born



This query returns the value of the born property of the Tom Hanks node.

In Cypher, labels, property keys, and variables are case-sensitive. Cypher keywords are not case-sensitive.

Neo4j best practices include:

* Name labels using **CamelCase**.
* Name property keys and variables using **camelCase**.
* User **UPPERCASE** for Cypher keywords.

Another way that you can filter queries is by using the WHERE clause, rather than specifying the property value inline with braces.

This query returns the same data as the previous query.

**cypher**

Copy to ClipboardRun in Sandbox

MATCH (p:Person)

WHERE p.name = 'Tom Hanks'

RETURN p.born



As you gain more experience with Cypher, you will find that using WHERE to filter your queries is very powerful because you can add more logic to your WHERE clause. Here is an example where we filter by two values for name.

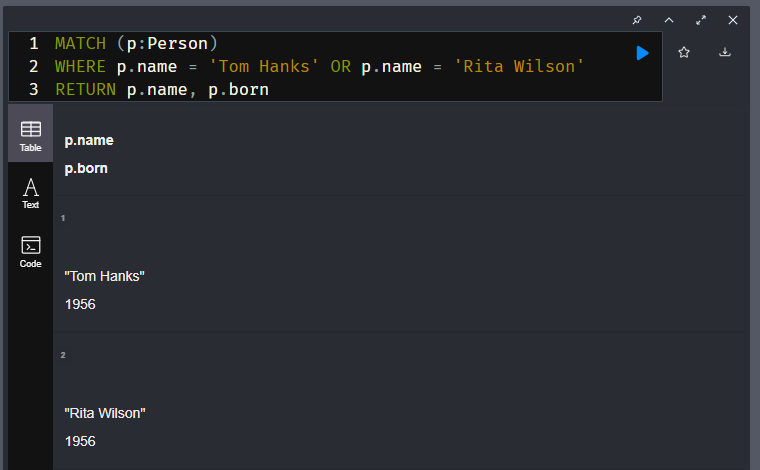
**cypher**

Copy to ClipboardRun in Sandbox

MATCH (p:Person)

WHERE p.name = 'Tom Hanks' OR p.name = 'Rita Wilson'

RETURN p.name, p.born



This query returns two names and their associated birth years.

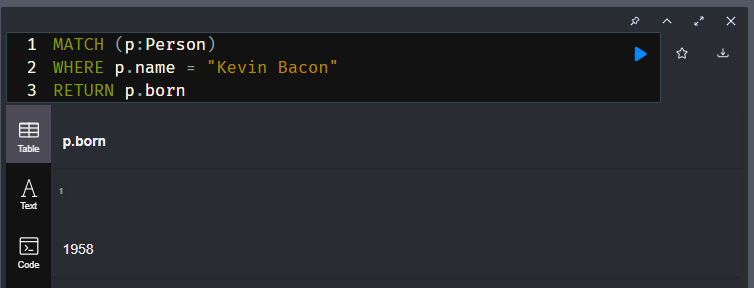
### Question(s) / Example(s) –

Retreives all the property names of the person label i.e returns all the names of the persons label

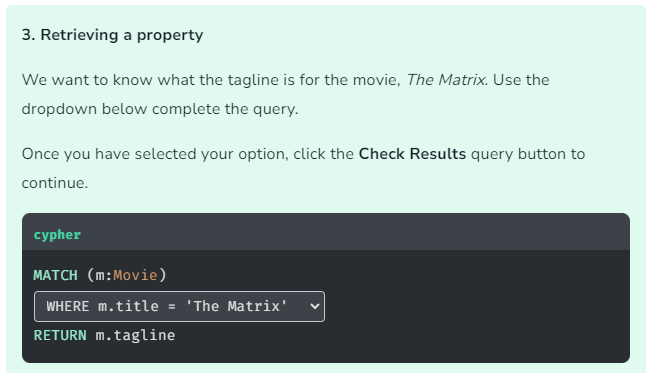


Another Example-

Retreives the year the person - Kevin Bacon was born in



Another Example-



### Main Point-

1. The **MATCH** clause is used to read data from neo4j.

Basically the **MATCH** clause is used to define the pattern in the graph that you would like to search for.

1. **WHERE** keyword can be used to filter the nodes retrieved during the execution of a **MATCH** clause.

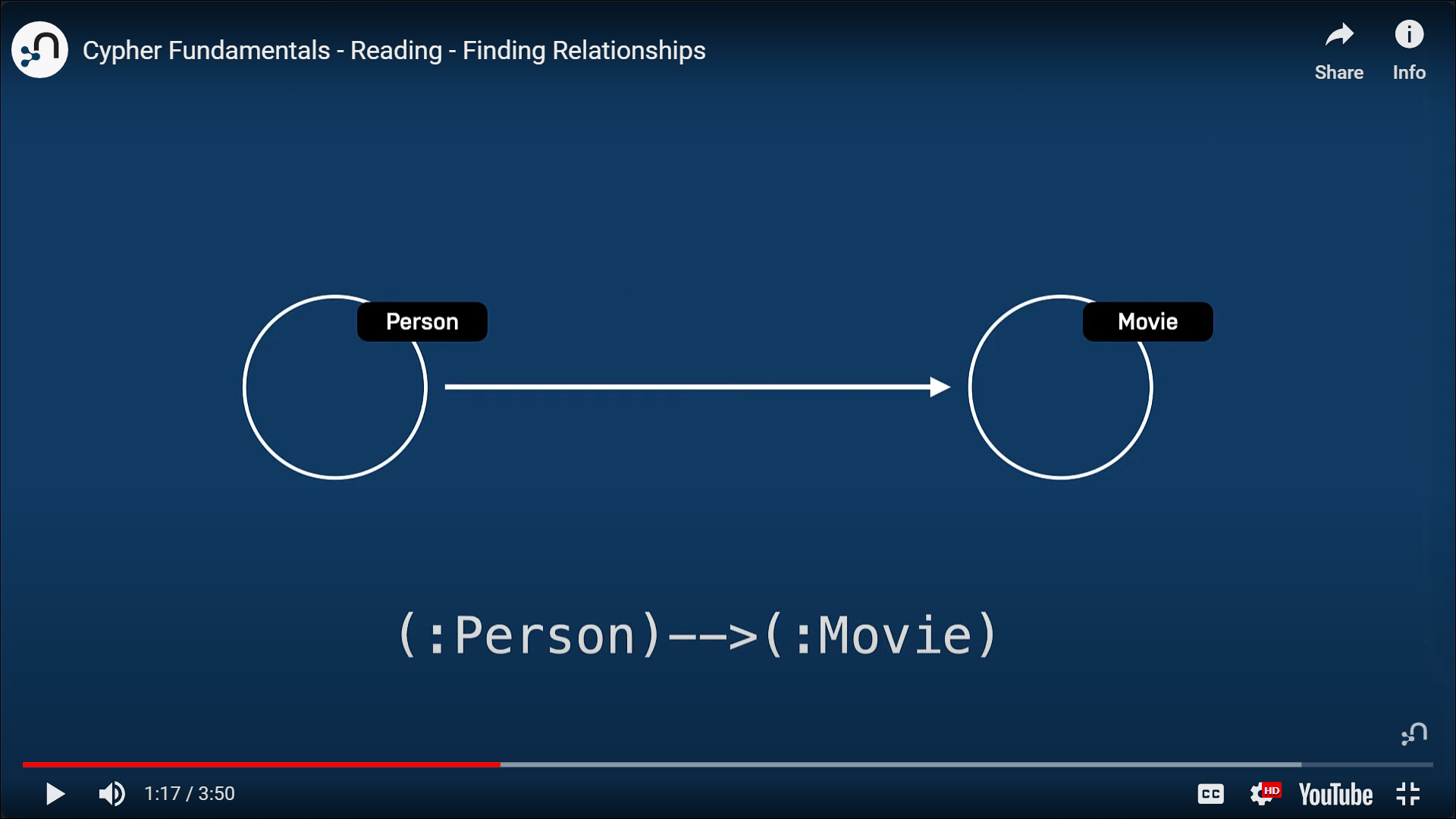
Basically Just like SQL, you use the **WHERE** clause to filter results in Cypher.

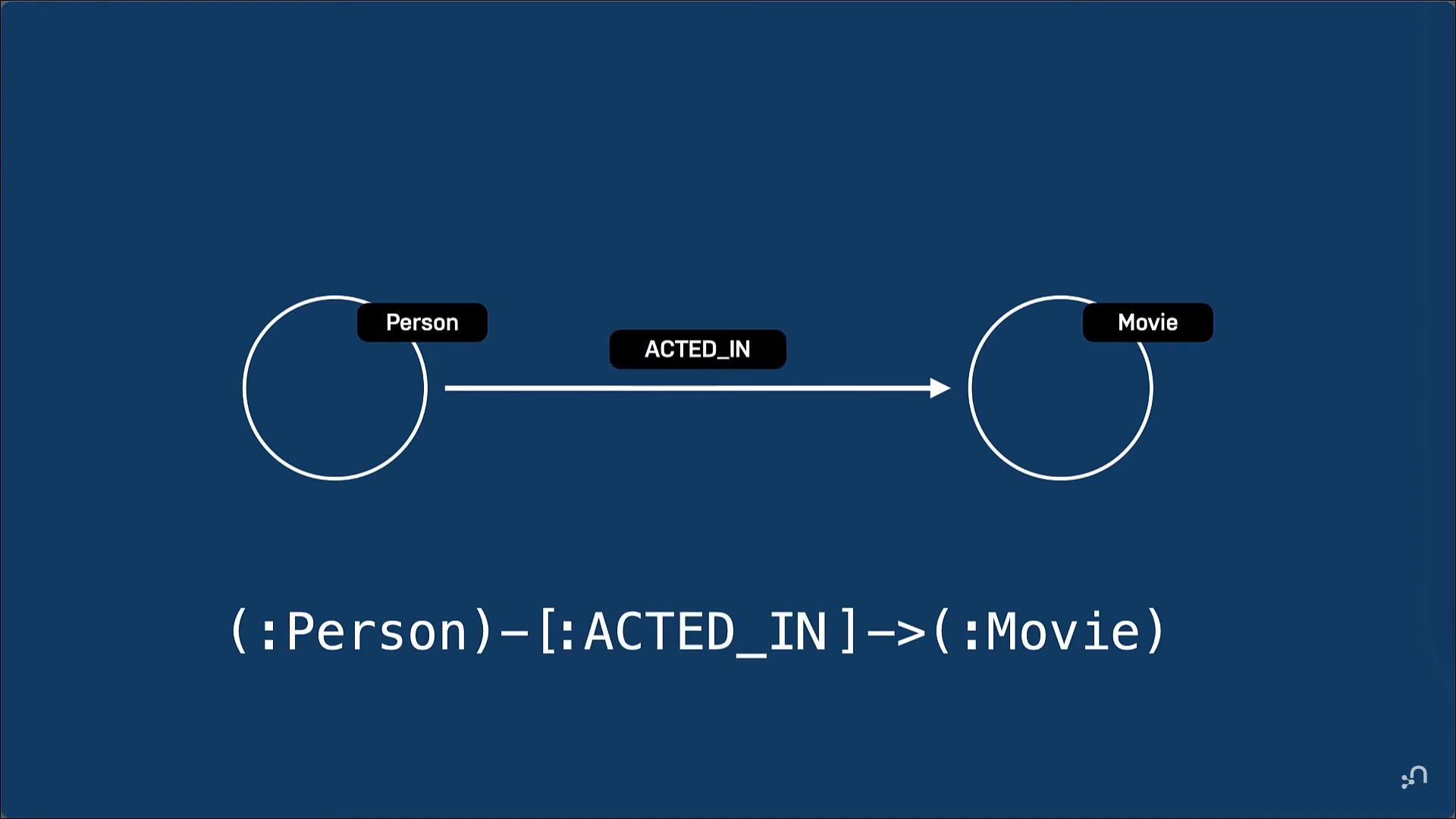
# Finding Relationships

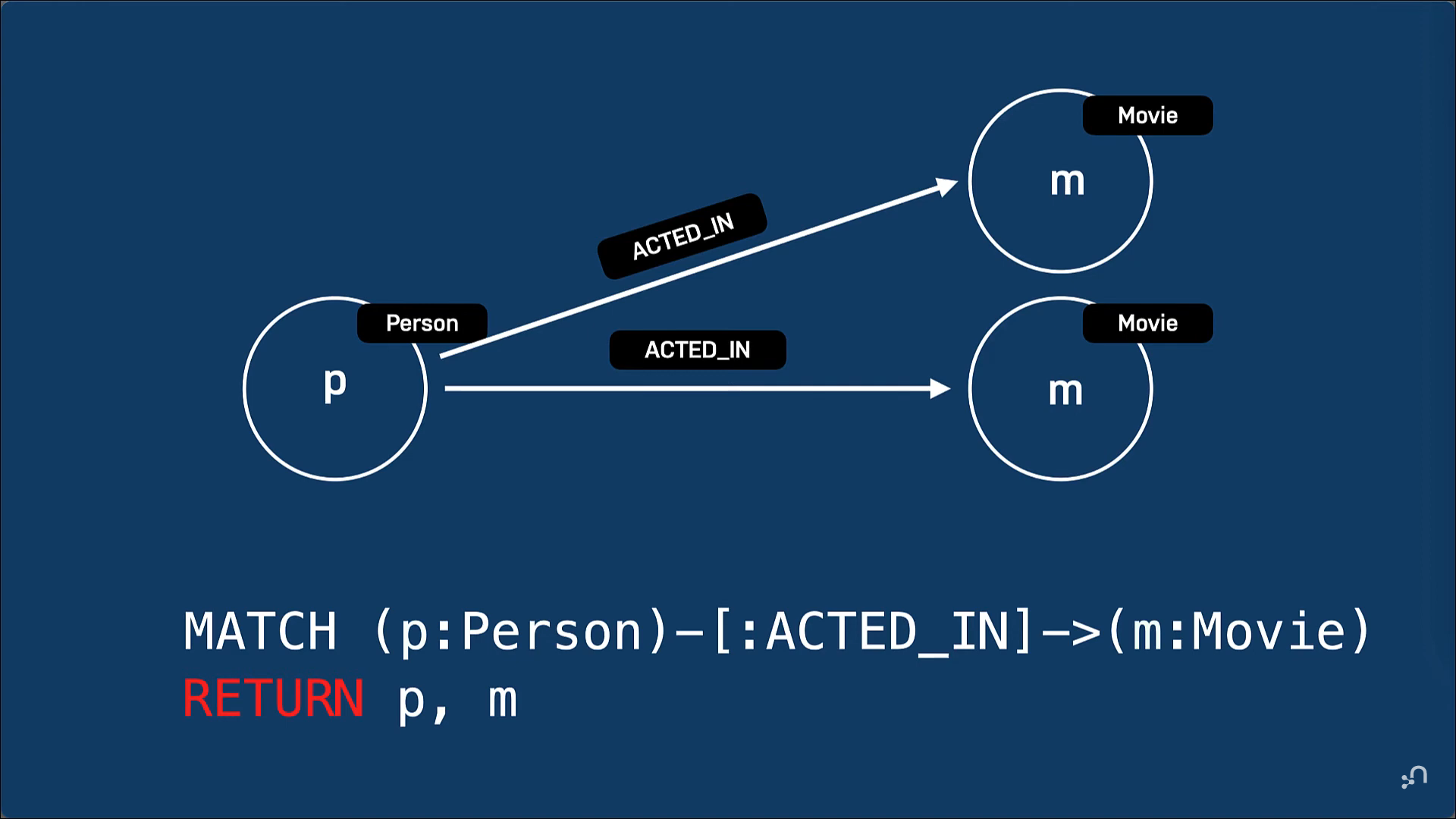
Youtube link: <https://youtu.be/aQ5hKd2DD0A>

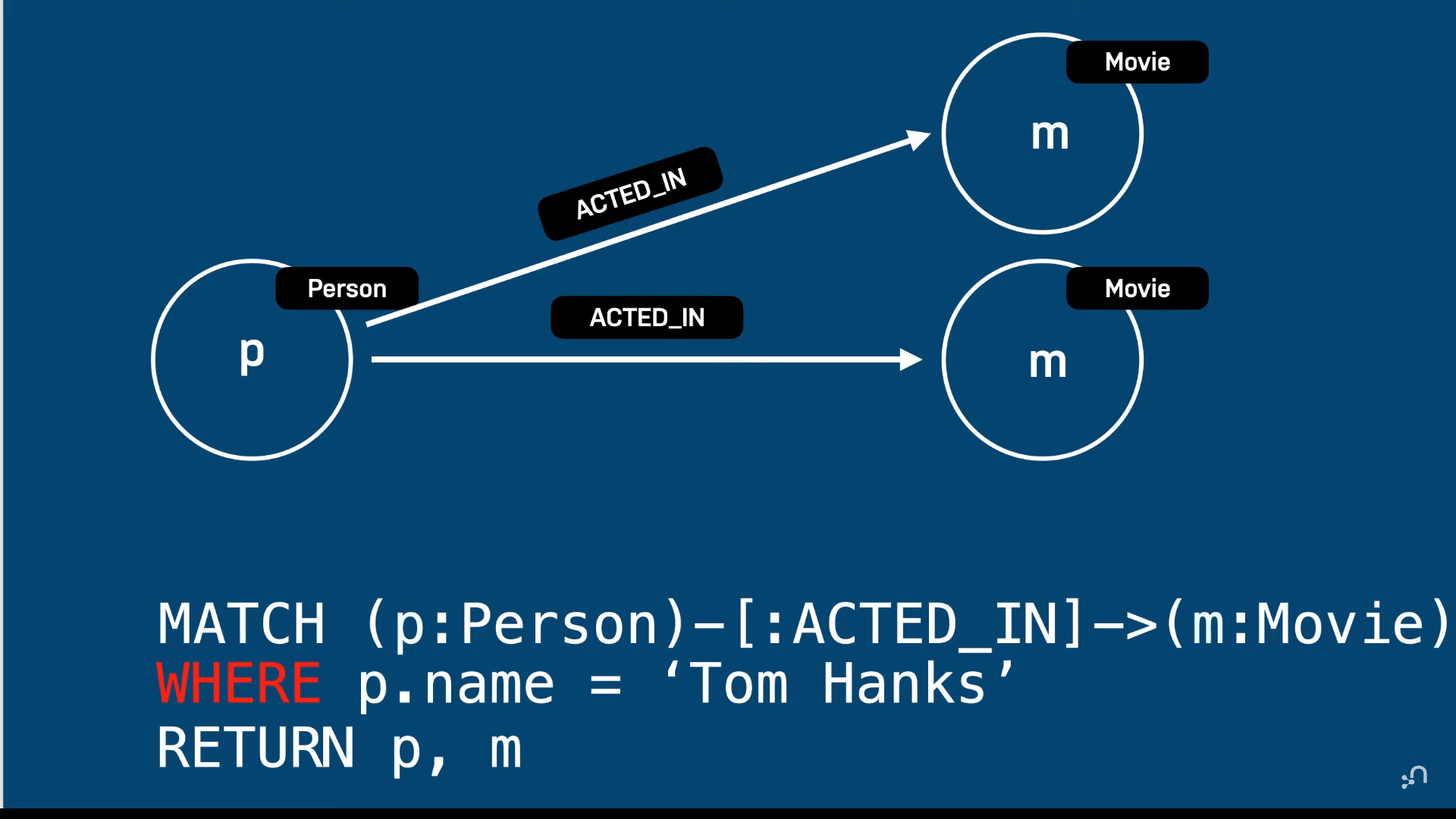
Transcript-

Sandbox neo4j lesson is there and contains screenshots









**OUR GOAL**

As a **movie fanatic**  
I would like to **find movies for a particular actor**  
So that I can **watch a movie this evening**

In the previous lesson, we used the MATCH clause to find the node in our database that represented *Tom Hanks*.

**cypher**

Copy to ClipboardRun in Sandbox

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

RETURN p



We can extend the pattern in the MATCH clause to *traverse* through all relationships with a type of *ACTED\_IN* to any node. Our domain model shows that the *ACTED\_IN* relationship goes in an outgoing direction from the *Person* node so we can add the direction in our pattern. We often refer to this as a **traversal**.

**cypher**

**Incomplete code**

MATCH (p:Person {name: 'Tom Hanks'})-[:ACTED\_IN]->()

Our data model dictates that the node at the other end of that relationship will be *Movie* node, so we don’t necessarily need to specify the *:Movie* label in the node - instead we will use the variable *m*.

**cypher**

Copy to ClipboardRun in Sandbox

MATCH (p:Person {name: 'Tom Hanks'})-[:ACTED\_IN]->(m)

RETURN m.title



This code returns the titles of all movies that *Tom Hanks* acted in.

If our graph had different labels, for example Television and *Movie* nodes this query would have returned all *Television* and *Movie* nodes that Tom Hanks acted in. That is, if we had multiple *types* of nodes at the end of the *ACTED\_IN* relationships in our graph, we could make sure that we only return movies.

**cypher**

Copy to ClipboardRun in Sandbox

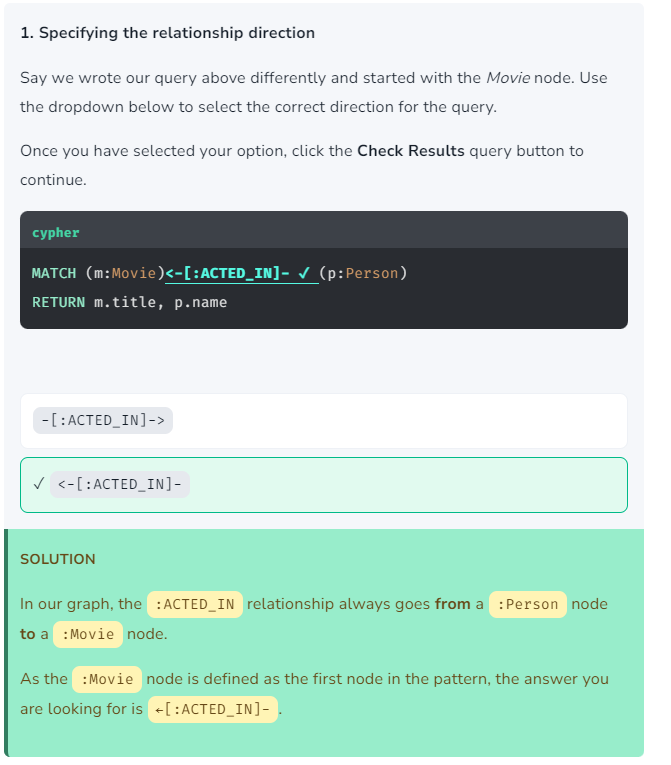
MATCH (p:Person {name: 'Tom Hanks'})-[:ACTED\_IN]->(m:Movie)

RETURN m.title

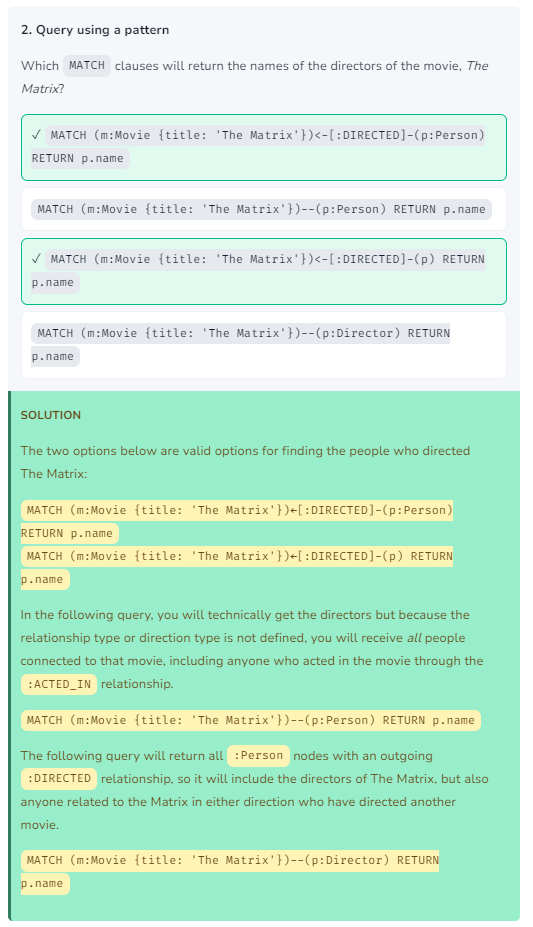


Because our graph only has *Movie* nodes that have incoming *ACTED\_IN* relationships, this query returns the exact same results as the previous query.

### Question(s) / Example(s) –

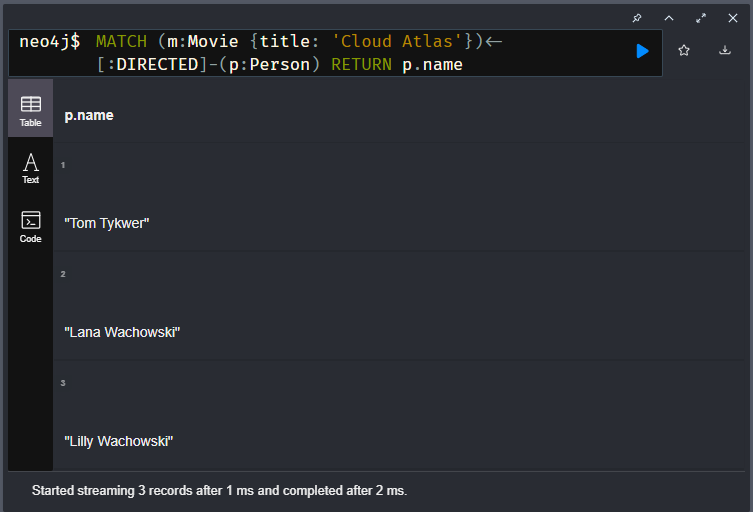


Another Example-



Another Example-

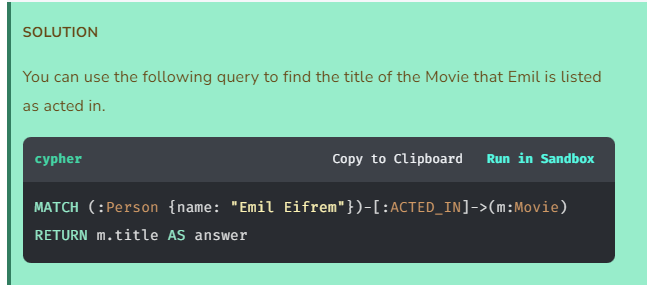
Finding the people who directed the movie Cloud Atlas



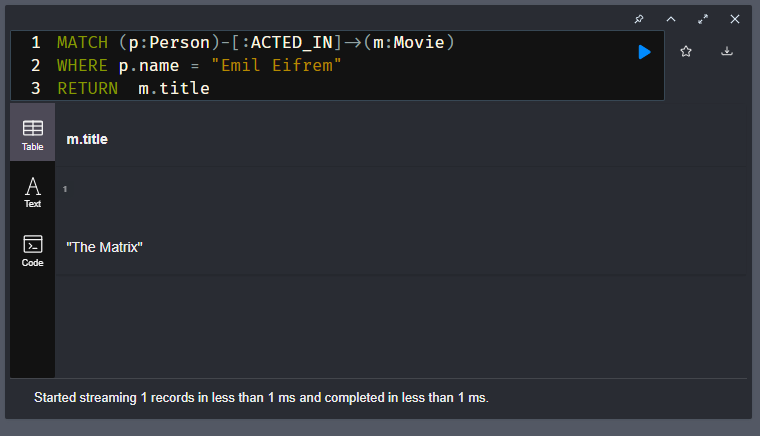
Or you can return the count of directors by-



Another Example-



Or



# Filtering Queries

Youtube link: <https://youtu.be/ZBbH5p-5Gt8>

Transcript-

Sandbox neo4j lesson is there and contains screenshots

Earlier, you learned that the WHERE clause is used to tell the query engine to filter what nodes are retrieved from the graph. In this lesson you will learn about some of the ways that you can filter your queries.

You have already learned how you can test equality for properties of a node and how you can use logical expressions to further filter what you want to retrieve.

For example, this query retrieves the Person nodes and Movie nodes where the person acted in a movie that was released in 2008 or 2009:

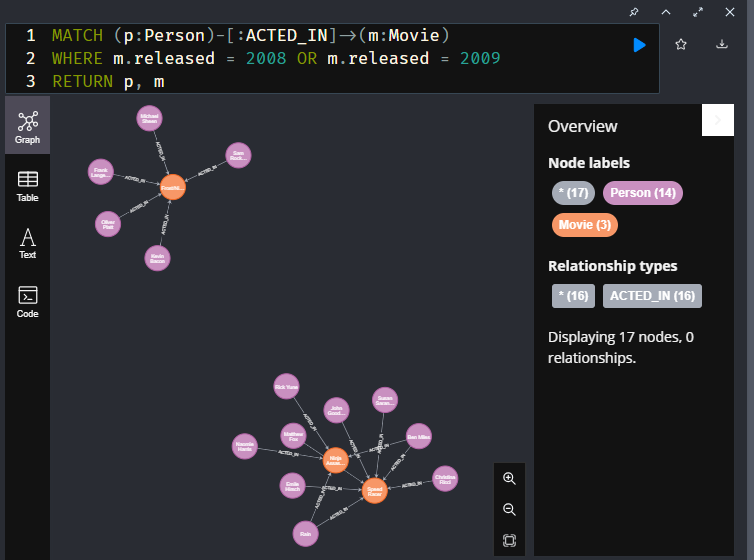
**cypher**

Copy to ClipboardRun in Sandbox

MATCH (p:Person)-[:ACTED\_IN]->(m:Movie)

WHERE m.released = 2008 OR m.released = 2009

RETURN p, m



### Filtering by node labels

You have already seen this type of query. It returns the names of all people who acted in the movie, The Matrix.

**cypher**

Copy to ClipboardRun in Sandbox

MATCH (p:Person)-[:ACTED\_IN]->(m:Movie)

WHERE m.title='The Matrix'

RETURN p.name



An alternative to this query is the following where we test the node labels in the WHERE clause:

**cypher**

Copy to ClipboardRun in Sandbox

MATCH (p)-[:ACTED\_IN]->(m)

WHERE p:Person AND m:Movie AND m.title='The Matrix'

RETURN p.name



Both queries execute the same way, but you may want to use one style of filtering over another in your code.

### Filtering using ranges

You can specify a range for filtering a query. Here we want to retrieve Person nodes of people who acted in movies released between 2000 and 2003:

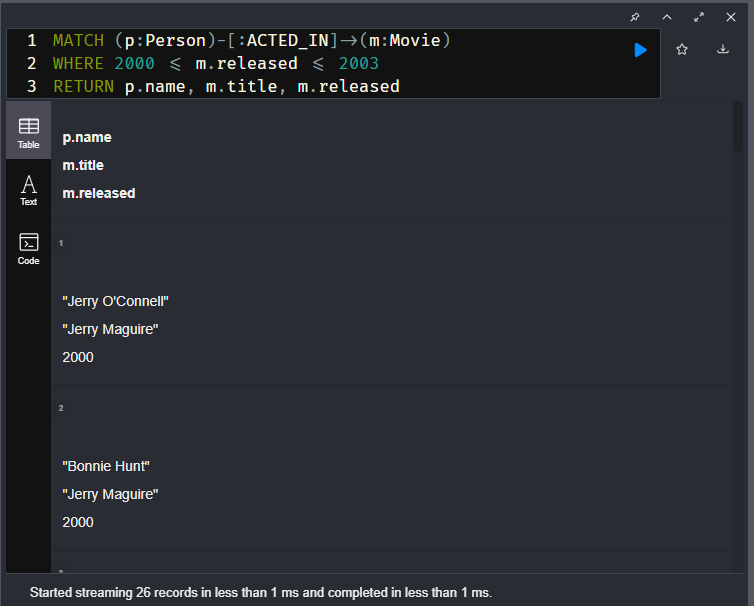
**cypher**

Copy to ClipboardRun in Sandbox

MATCH (p:Person)-[:ACTED\_IN]->(m:Movie)

WHERE 2000 <= m.released <= 2003

RETURN p.name, m.title, m.released



### Filtering by existence of a property

Recall that by default, there is no requirement that a node or relationship has a given property. Here is an example of a query where we only want to return Movie nodes where Jack Nicholson acted in the movie, and the movie has the tagline property.

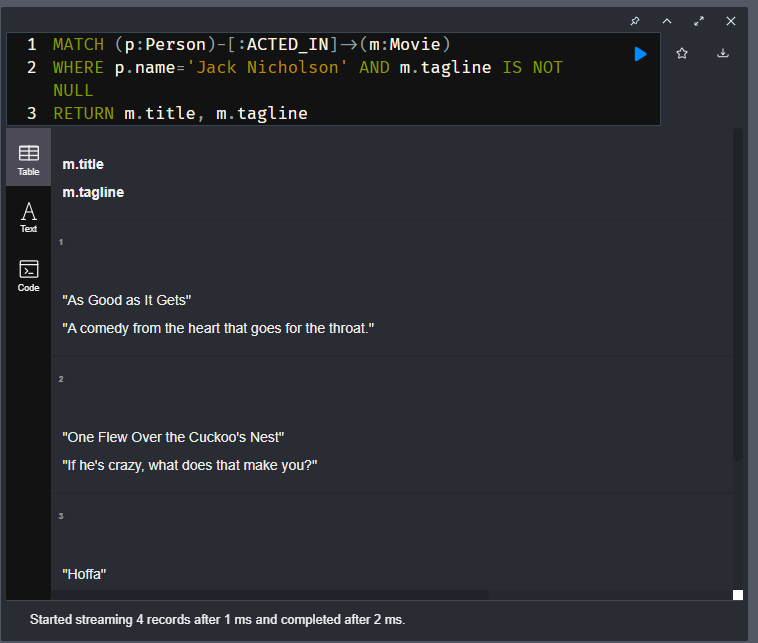
**cypher**

Copy to ClipboardRun in Sandbox

MATCH (p:Person)-[:ACTED\_IN]->(m:Movie)

WHERE p.name='Jack Nicholson' AND m.tagline IS NOT NULL

RETURN m.title, m.tagline



### Filtering by partial strings

Cypher has a set of string-related keywords that you can use in your WHERE clauses to test string property values. You can specify STARTS WITH, ENDS WITH, and CONTAINS.

For example, to find all actors in the graph whose first name is Michael, you would write:

**cypher**

Copy to ClipboardRun in Sandbox

MATCH (p:Person)-[:ACTED\_IN]->()

WHERE p.name STARTS WITH 'Michael'

RETURN p.name



String tests are case-sensitive so you may need to use the toLower() or toUpper() functions to ensure the test yields the correct results. For example:

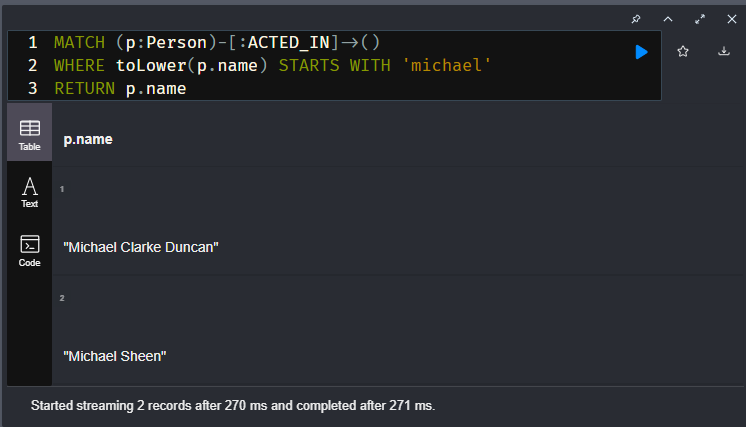
**cypher**

Copy to ClipboardRun in Sandbox

MATCH (p:Person)-[:ACTED\_IN]->()

WHERE toLower(p.name) STARTS WITH 'michael'

RETURN p.name



### Filtering by patterns in the graph

Suppose you wanted to find all people who wrote a movie but did not direct that same movie. Here is how you would perform the query:

**cypher**

Copy to ClipboardRun in Sandbox

MATCH (p:Person)-[:WROTE]->(m:Movie)

WHERE NOT exists( (p)-[:DIRECTED]->(m) )

RETURN p.name, m.title



### Filtering using lists

If you have a set of values you want to test with, you can place them in a list or you can test with an existing list in the graph. A Cypher list is a comma-separated set of values within square brackets.

You can define the list in the WHERE clause. During the query, the graph engine will compare each property with the values IN the list. You can place either numeric or string values in the list, but typically, elements of the list are of the same type of data. If you are testing with a property of a string type, then all the elements of the list will be strings.

In this example, we only want to retrieve Person nodes of people born in 1965, 1970, or 1975:

**cypher**

Copy to ClipboardRun in Sandbox

MATCH (p:Person)

WHERE p.born IN [1965, 1970, 1975]

RETURN p.name, p.born



You can also compare a value to an existing list in the graph.

We know that the :ACTED\_IN relationship has a property, roles that contains the list of roles an actor had in a particular movie they acted in. Here is the query we write to return the name of the actor who played Neo in the movie The Matrix:

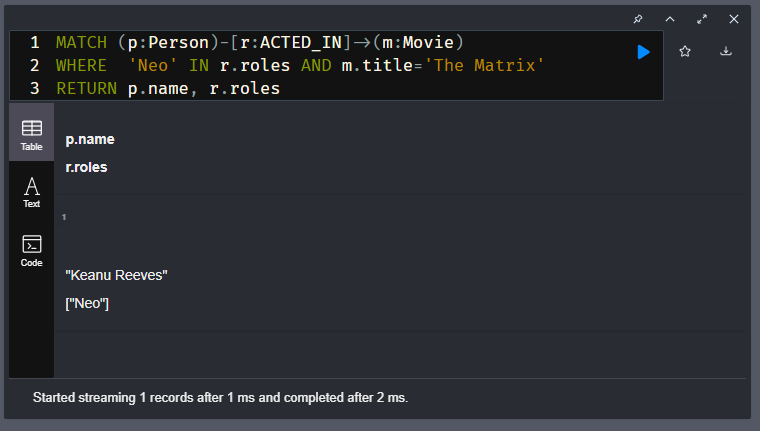
**cypher**

Copy to ClipboardRun in Sandbox

MATCH (p:Person)-[r:ACTED\_IN]->(m:Movie)

WHERE 'Neo' IN r.roles AND m.title='The Matrix'

RETURN p.name, r.roles



### What properties does a node or relationship have?

The properties for a node with a given label need not be the same. One way you can discover the properties for a node is to use the keys() function. This function returns a list of all property keys for a node.

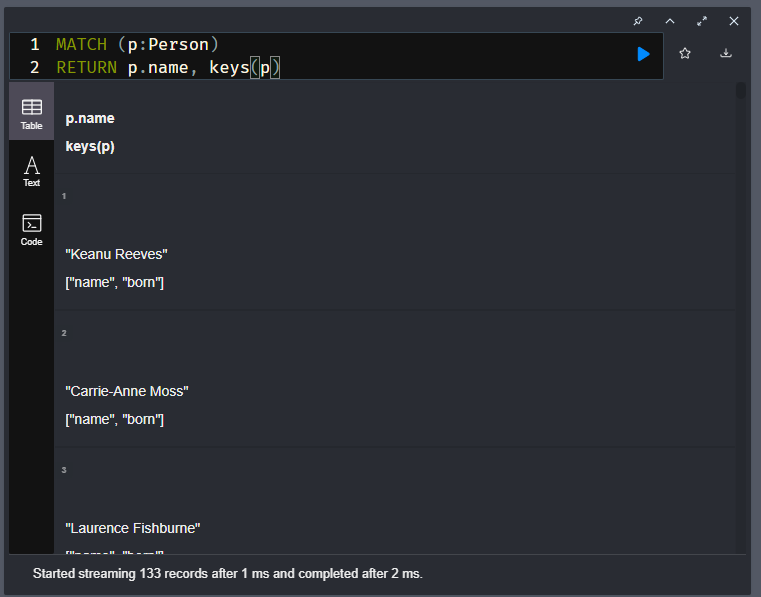
Discover the keys for the Person nodes in the graph by running this code:

**cypher**

Copy to ClipboardRun in Sandbox

MATCH (p:Person)

RETURN p.name, keys(p)



The results returned for each row include the name of the person, followed by the list of property keys for that node. If you scroll down in the result pane, you will notice that some Person nodes do not have a born property.

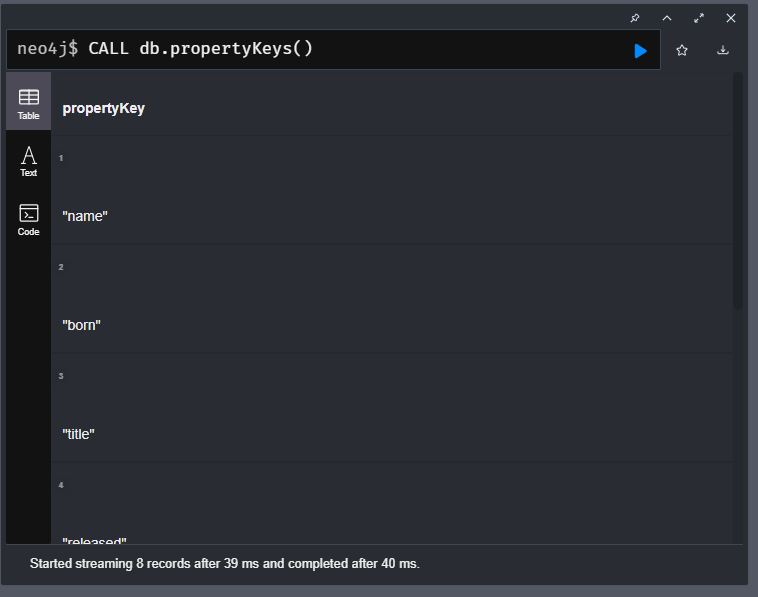
### What properties exist in the graph?

More generally, you can run this code to return all the property keys defined in the graph.

**cypher**

Copy to ClipboardRun in Sandbox

CALL db.propertyKeys()



Note that a property key remains in the graph, once it has been defined, even if there are currently no nodes or relationships that use that property key.

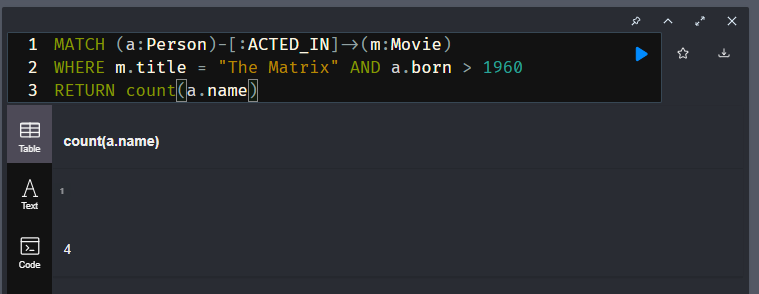
### Question(s) / Example(s)-



Another Example-



Or



# Module 2: Writing Data to Neo4j

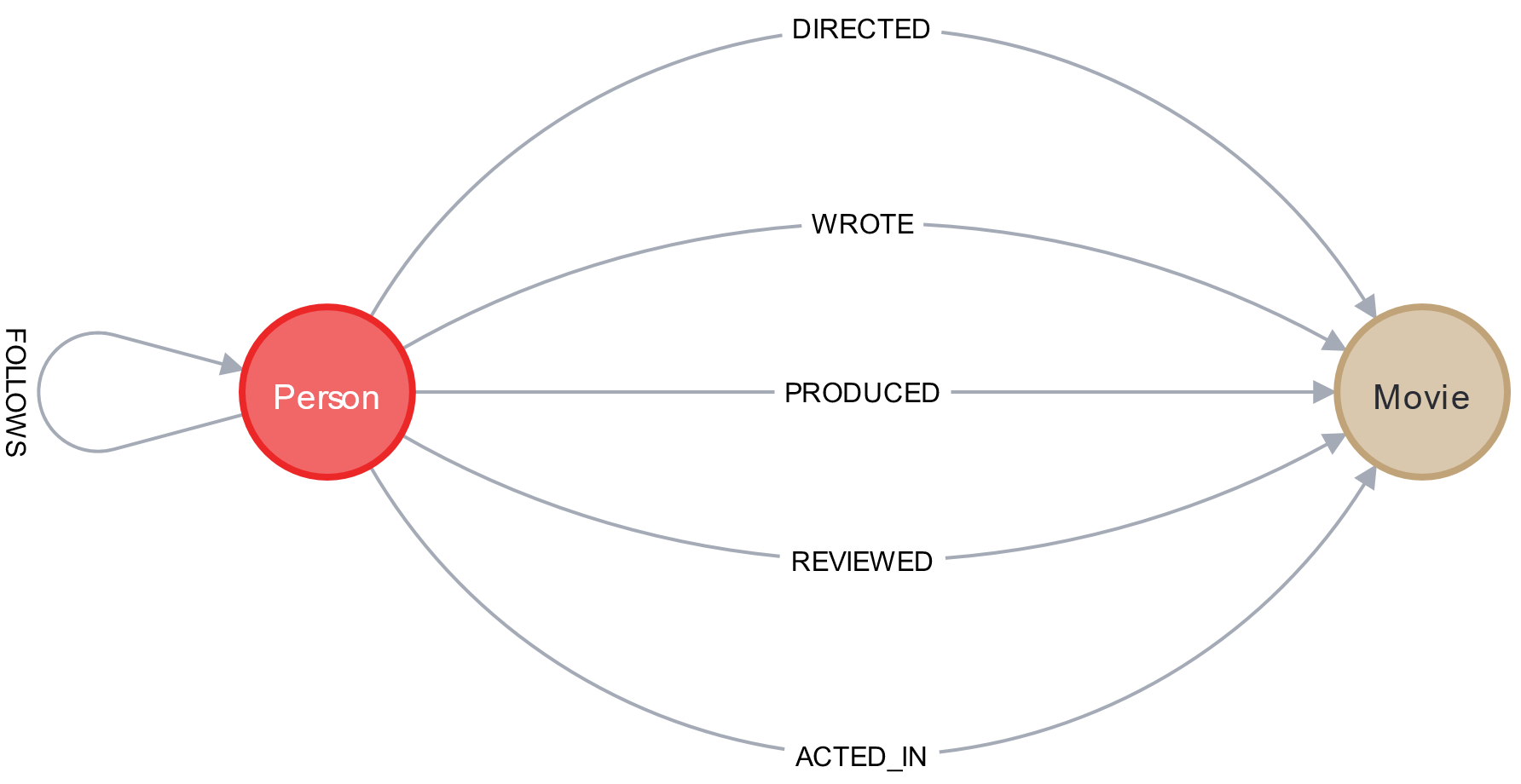
In this module you will learn how to update the graph using Cypher and the Movies example dataset.

You will learn to:

* Use MERGE to create nodes in the graph.
* Use MERGE to create relationships in the graph.
* Create, update and remove properties for nodes and relationships in the graph.
* Perform conditional MERGE processing, depending on what is in the graph.
* Delete nodes and relationships from the graph.

## Domain model for this course

Again, here is the domain model and how it is represented in our graph:



# Creating Nodes

Youtube link: <https://youtu.be/fgoz4eMQHOk>

Transcript-

Sandbox neo4j lesson is there and contains screenshots

## Creating nodes

In this lesson you will learn how to write Cypher code to create nodes in the graph.

Using the Movies data model, you will create and execute Cypher code to create actors and movies in our graph.

We use the MERGE keyword to create a pattern in the database.

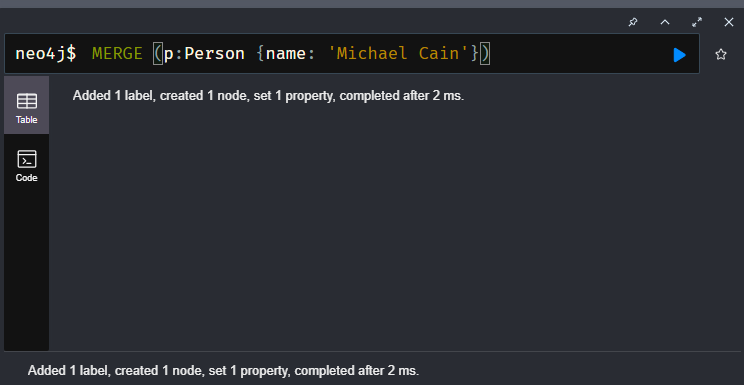
After the MERGE keyword, we specify the pattern that we want to create. Usually this will be a single node or a relationship between two nodes.

Suppose we want to create a node to represent Michael Caine. Run this Cypher code to create the node.

**cypher**

Copy to ClipboardRun in Sandbox

MERGE (p:Person {name: 'Michael Caine'})



It creates a single node in the graph. Note that when you use MERGE to create a node, you must specify at least one property that will be the unique primary key for the node.

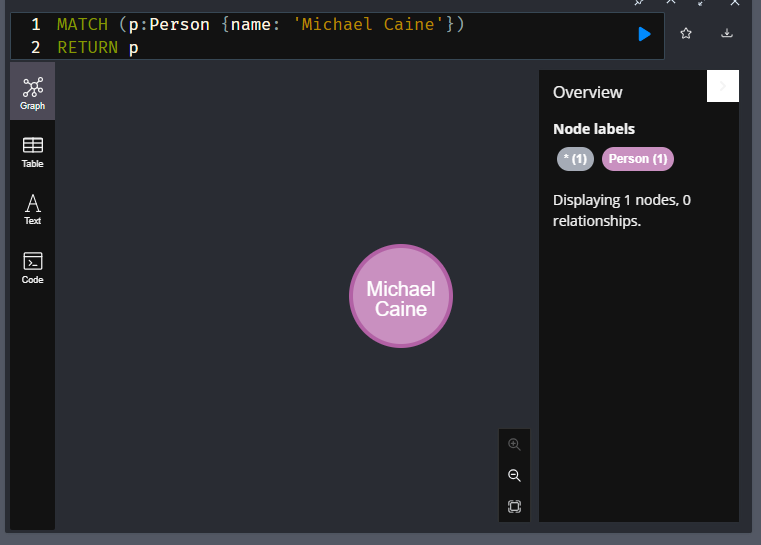
Verify that the node was created.

**cypher**

Copy to ClipboardRun in Sandbox

MATCH (p:Person {name: 'Michael Caine'})

RETURN p





### Executing multiple Cypher clauses

We can also chain multiple MERGE clauses together within a single Cypher code block.

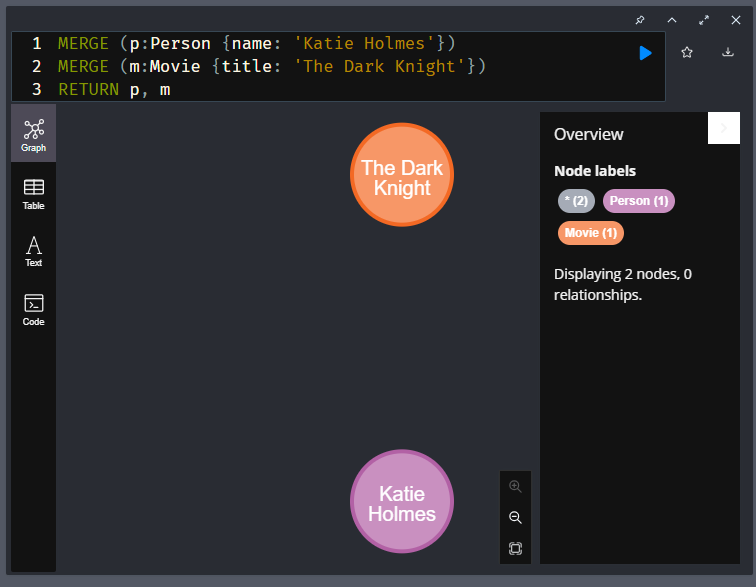
**cypher**

Copy to ClipboardRun in Sandbox

MERGE (p:Person {name: 'Katie Holmes'})

MERGE (m:Movie {title: 'The Dark Knight'})

RETURN p, m



This code creates two nodes, each with a primary key property. Because we have specified the variables p and m, we can use them in the code to return the created nodes.

### Using **CREATE** instead of **MERGE** to create nodes

Cypher has a CREATE clause you can use for creating nodes. The benefit of using CREATE is that it does not look up the primary key before adding the node. You can use CREATE if you are sure your data is clean and you want greater speed during import. We use MERGE in this training because it eliminates duplication of nodes.

### Main Point-

1. MERGE and CREATE are used to create nodes or data
2. When merging a node you must specify the label for the node and the name and value of the properties that uniquely identify the node.

# Creating Relationships

Youtube link: <https://youtu.be/9npLX5us1DU>

Transcript-

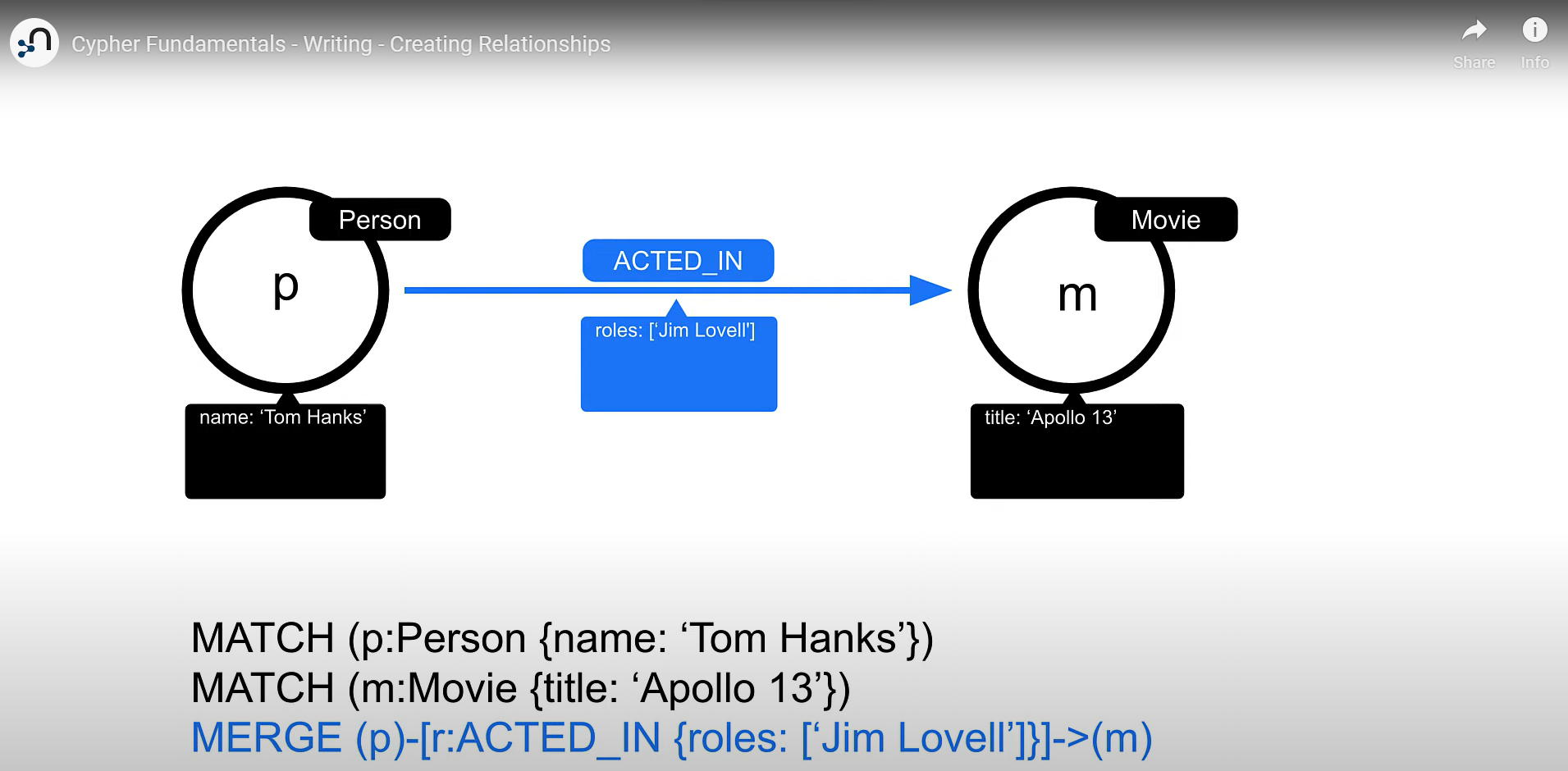
Sandbox neo4j lesson is there and contains screenshots

## Creating a relationship between two nodes

In this lesson you will learn how to write Cypher clauses to create relationships between existing nodes in the graph.

Just like you can use MERGE to create nodes in the graph, you use MERGE to create relationships between two nodes. First you must have references to the two nodes you will be creating the relationship for. When you create a relationship between two nodes, it must have:

* Type
* Direction



For example, if the Person and Movie nodes both already exist, we can find them using a MATCH clause before creating the relationship between them.

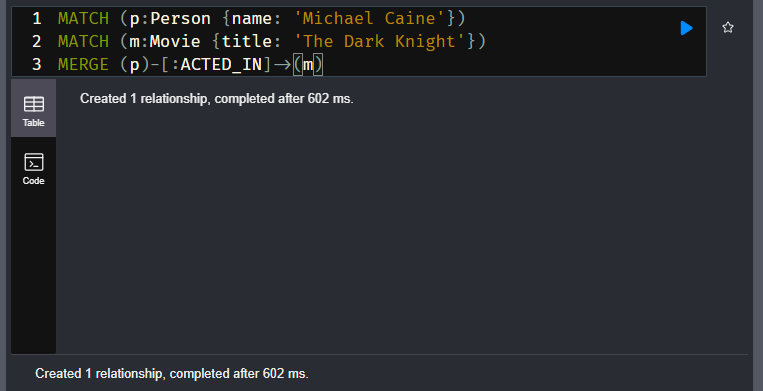
**cypher**

Copy to ClipboardRun in Sandbox

MATCH (p:Person {name: 'Michael Caine'})

MATCH (m:Movie {title: 'The Dark Knight'})

MERGE (p)-[:ACTED\_IN]->(m)



Here we find the two nodes that we want to create the relationship between. Then we use the reference to the found nodes to create the ACTED\_IN relationship.

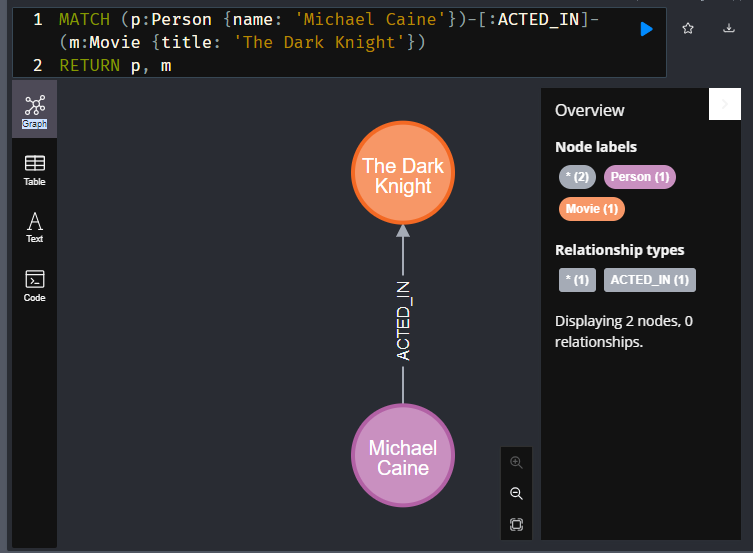
We can confirm that this relationship exists as follows:

**cypher**

Copy to ClipboardRun in Sandbox

MATCH (p:Person {name: 'Michael Caine'})-[:ACTED\_IN]-(m:Movie {title: 'The Dark Knight'})

RETURN p, m



By default, in Neo4j Browser, the visualization connects nodes that have relationships between them.

Notice also that you need not specify direction in the MATCH pattern since the query engine will look for all nodes that are connected, regardless of the direction of the relationship.

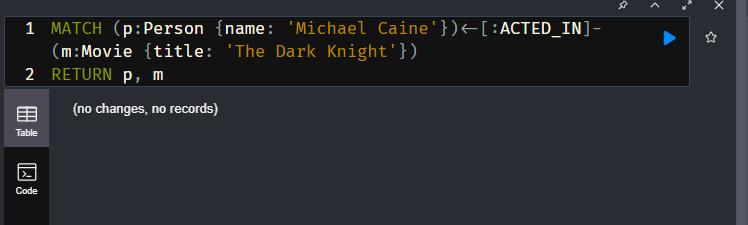
For example, if we specified this relationship pattern:

**cypher**

Copy to ClipboardRun in Sandbox

MATCH (p:Person {name: 'Michael Caine'})<-[:ACTED\_IN]-(m:Movie {title: 'The Dark Knight'})

RETURN p, m



This query returns no nodes since there are no nodes with the ACTED\_IN relationship to Person nodes in the graph.

### Creating nodes and relationships using multiple clauses

We can also chain multiple MERGE clauses together within a single Cypher code block.

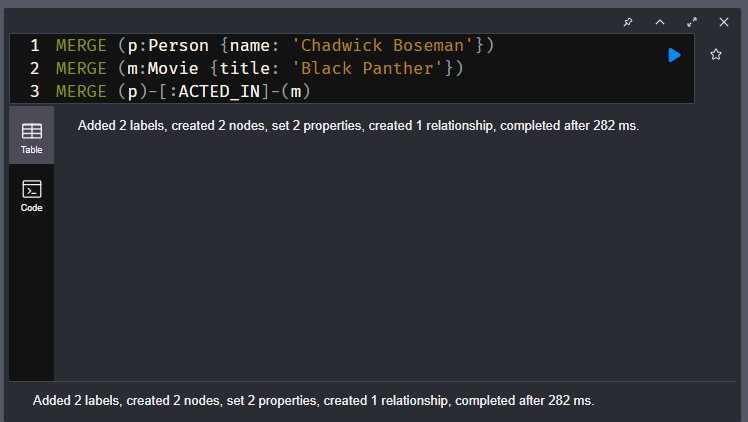
**cypher**

Copy to ClipboardRun in Sandbox

MERGE (p:Person {name: 'Chadwick Boseman'})

MERGE (m:Movie {title: 'Black Panther'})

MERGE (p)-[:ACTED\_IN]-(m)



This code creates two nodes and a relationship between them. Because we have specified the variables p and m, we can use them in the code to create the relationship between the two nodes.

Note that in this MERGE clause where we create the relationships, we did not specify the direction of the relationship. By default, if you do not specify the direction when you create the relationship, it will always be assumed left-to-right.

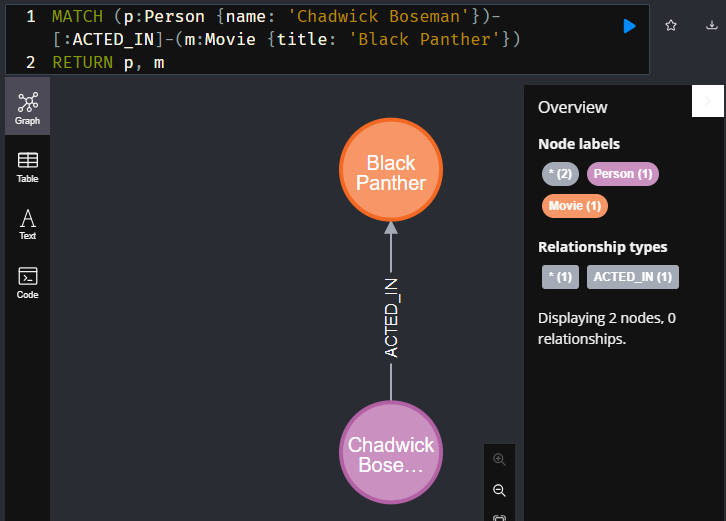
We can confirm that this relationship exists as follows:

**cypher**

Copy to ClipboardRun in Sandbox

MATCH (p:Person {name: 'Chadwick Boseman'})-[:ACTED\_IN]-(m:Movie {title: 'Black Panther'})

RETURN p, m



### Using **MERGE** to create nodes and a relationship in single clause

What MERGE does is create the node or relationship if it does not exist in the graph.

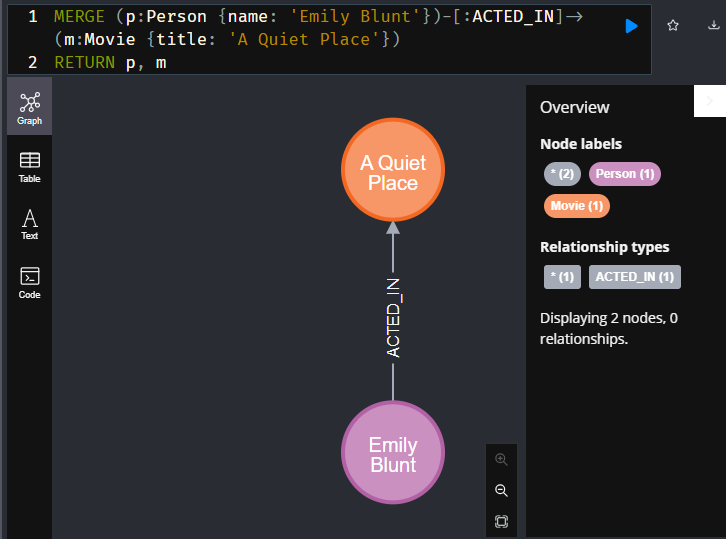
This code successfully creates the nodes and relationship:

**cypher**

Copy to ClipboardRun in Sandbox

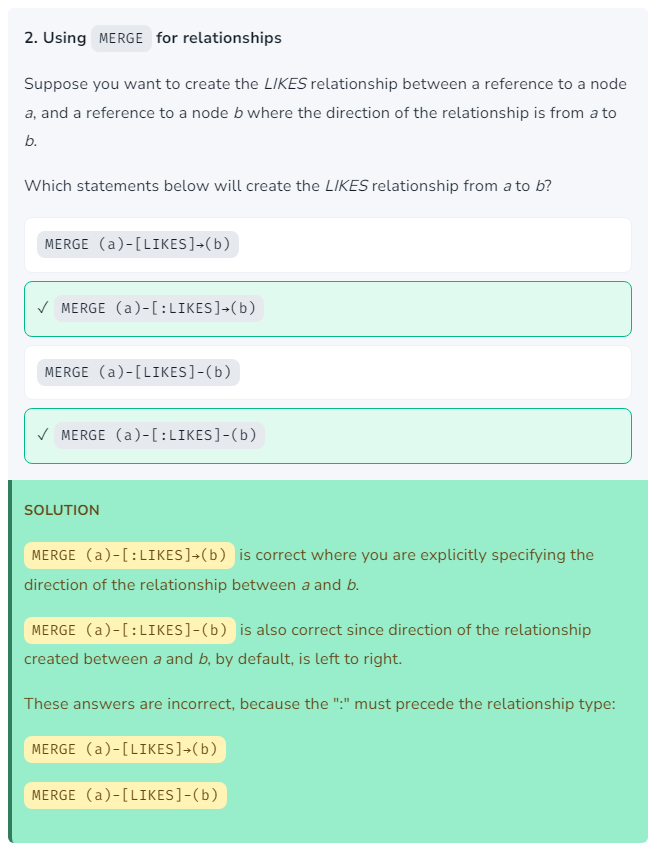
MERGE (p:Person {name: 'Emily Blunt'})-[:ACTED\_IN]->(m:Movie {title: 'A Quiet Place'})

RETURN p, m



You can execute this Cypher code multiple times and it will not create any new nodes or relationships.

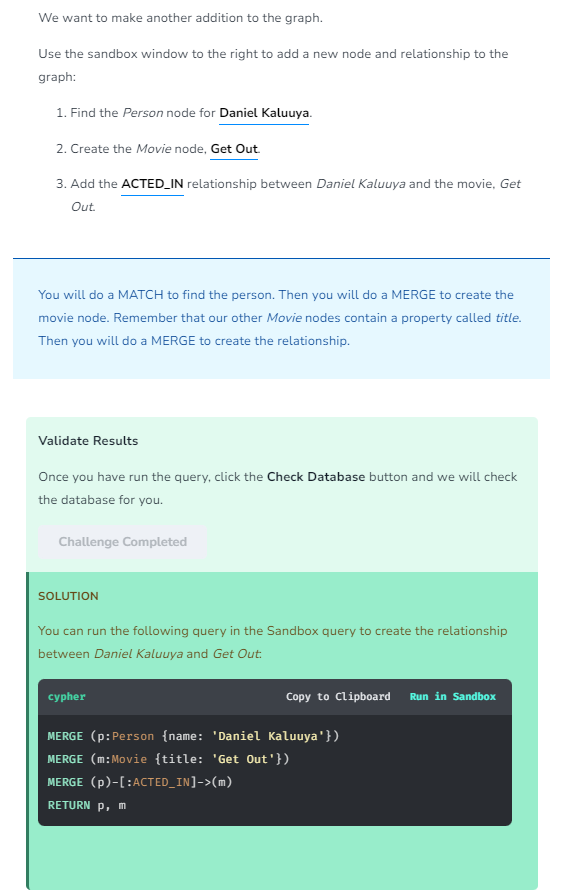
### Question(s) / Example(s)-



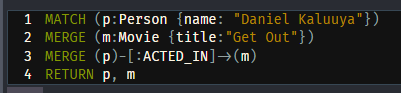
Another Example-



Another Example-



Alternate way for the same problem (not sure if correct)-



## Main Points-

1. **MERGE** is used to create a relationship.

You use the **MERGE** clause when you have a reference to the from node and the to node. It prevents you from creating duplicate relationships between nodes.

1. If you do not specify the direction when you create the relationship, it will always be assumed left-to-right

# Updating Properties

Youtube link: <https://youtu.be/TUubik76iZ4>

Transcript-

Sandbox neo4j lesson is there and contains screenshots

## Updating properties

Thus far, you have learned how to create nodes with MERGE where you specify the primary key property for the node. You can add, modify, or remove properties from nodes and relationships.

In this lesson you will learn how to write Cypher code to update properties of nodes and relationships.

### Adding properties for a node or relationship

There are two ways that you can set a property for a node or relationship.

#### 1. Inline as part of the **MERGE** clause

You have already seen how to create the primary key property for a node. You can also set a property for a relationship inline as follows:

**cypher**

Copy to ClipboardRun in Sandbox

MERGE (p:Person {name: 'Michael Caine'})

MERGE (m:Movie {title: 'Batman Begins'})

MERGE (p)-[:ACTED\_IN {roles: ['Alfred Penny']}]->(m)

RETURN p,m



In this code, the actor, Michael Caine exists but the movie, Batman Begins does not. We find the Person node and we create the Movie node. Then, we create the ACTED\_IN relationship between the Michael Caine node and the newly-created Batman Begins node. And we set the roles property for this relationship to an array of values - containing one value, Alfred Penny. Notice that for inline property setting, we use the JSON-style of adding the property key/value pairs in braces { .. }, just like we did when we specified the property for the node.

#### 2. Using the **SET** keyword for a reference to a node or relationship

We also have the option to use the SET keyword for setting a property value. In the context of particular MERGE or MATCH clause where you have defined a variable to reference the node or relationship, you can set property values.

**cypher**

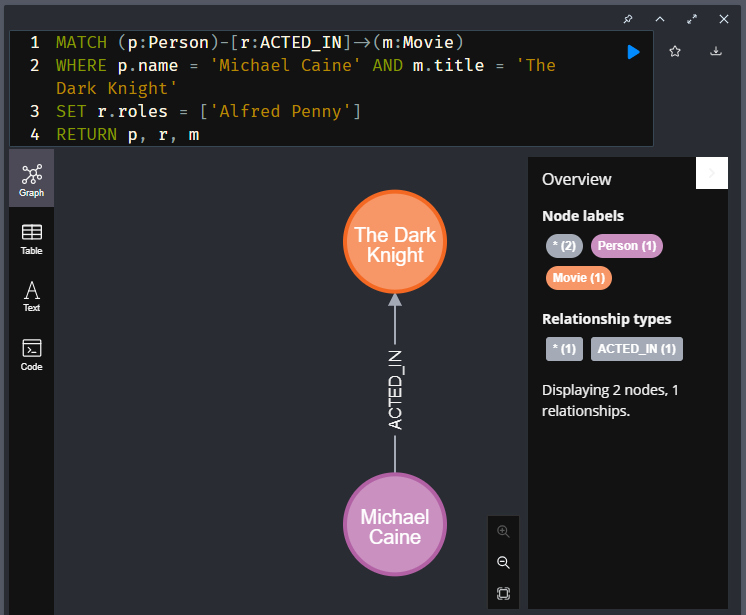
Copy to ClipboardRun in Sandbox

MATCH (p:Person)-[r:ACTED\_IN]->(m:Movie)

WHERE p.name = 'Michael Caine' AND m.title = 'The Dark Knight'

SET r.roles = ['Alfred Penny']

RETURN p, r, m



#### Setting multiple properties

If you need to set multiple properties, you separate them with a comma (,). For example:

**cypher**

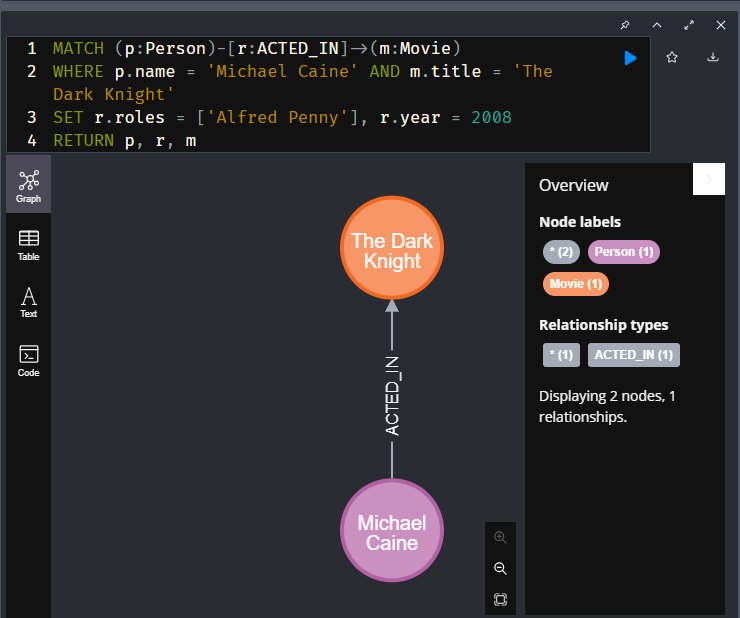
Copy to ClipboardRun in Sandbox

MATCH (p:Person)-[r:ACTED\_IN]->(m:Movie)

WHERE p.name = 'Michael Caine' AND m.title = 'The Dark Knight'

SET r.roles = ['Alfred Penny'], r.year = 2008

RETURN p, r, m



### Updating properties

If you have a reference to a node or relationship, you can also use SET to modify the property. For example, if we wanted to modify Michael Caine’s role to be something different, we could do the following:

**cypher**

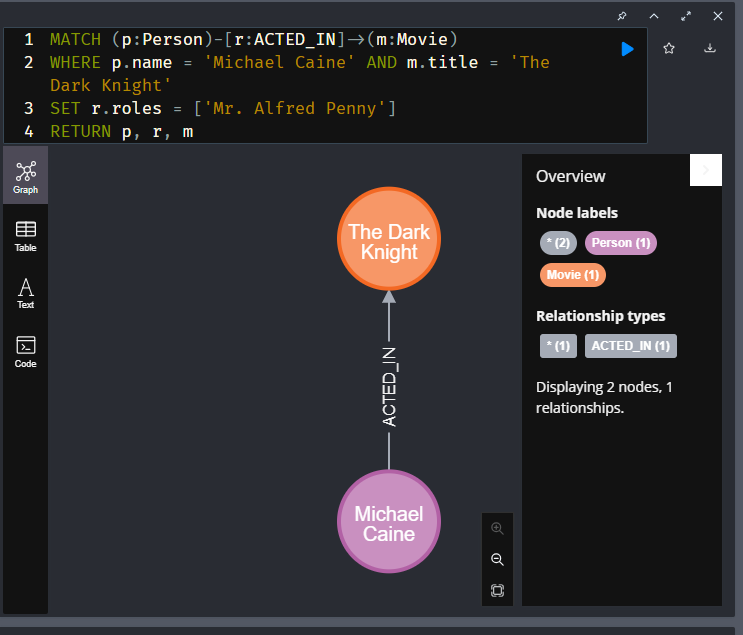
Copy to ClipboardRun in Sandbox

MATCH (p:Person)-[r:ACTED\_IN]->(m:Movie)

WHERE p.name = 'Michael Caine' AND m.title = 'The Dark Knight'

SET r.roles = ['Mr. Alfred Penny']

RETURN p, r, m



### Removing properties

You can remove or delete a property from a node or relationship by using the REMOVE keyword, or setting the property to null.

Here we remove the roles property of this relationship:

**cypher**

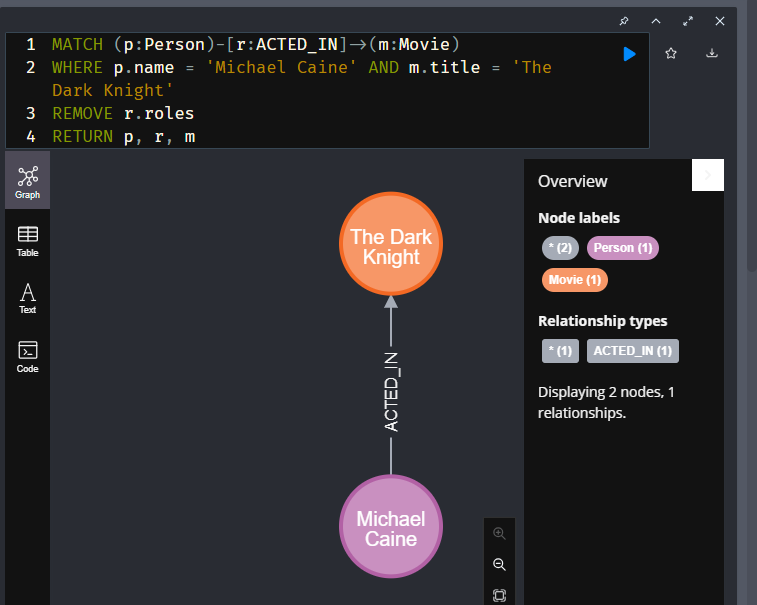
Copy to ClipboardRun in Sandbox

MATCH (p:Person)-[r:ACTED\_IN]->(m:Movie)

WHERE p.name = 'Michael Caine' AND m.title = 'The Dark Knight'

REMOVE r.roles

RETURN p, r, m



Here we remove the born property from an actor:

**cypher**

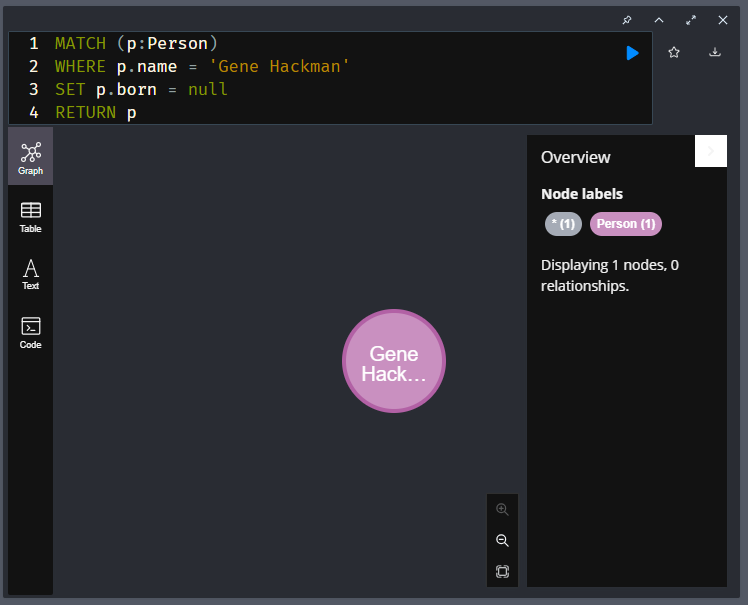
Copy to ClipboardRun in Sandbox

MATCH (p:Person)

WHERE p.name = 'Gene Hackman'

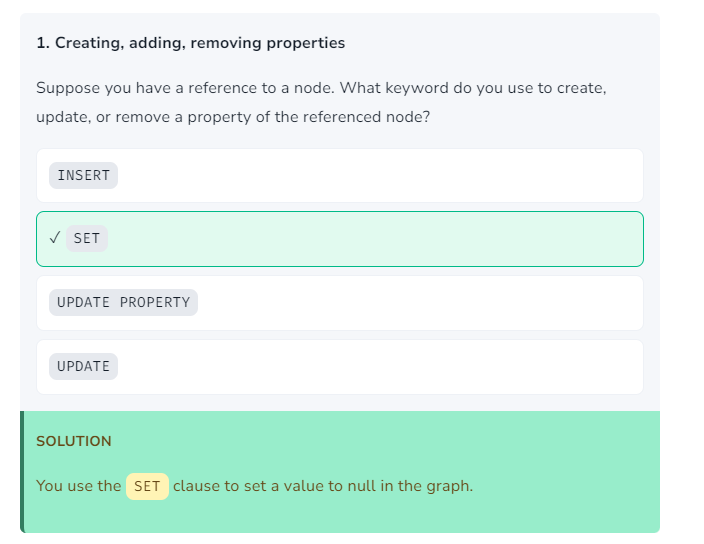
SET p.born = null

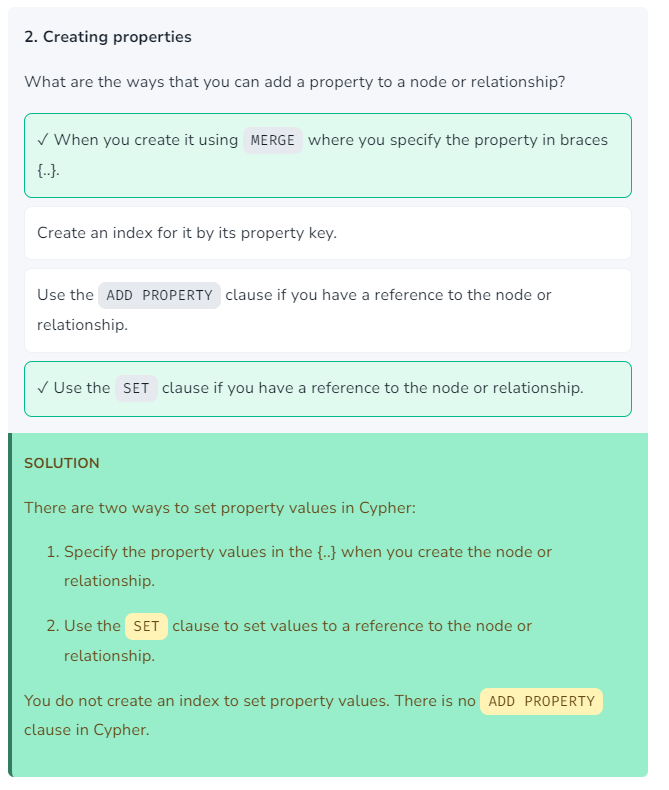
RETURN p

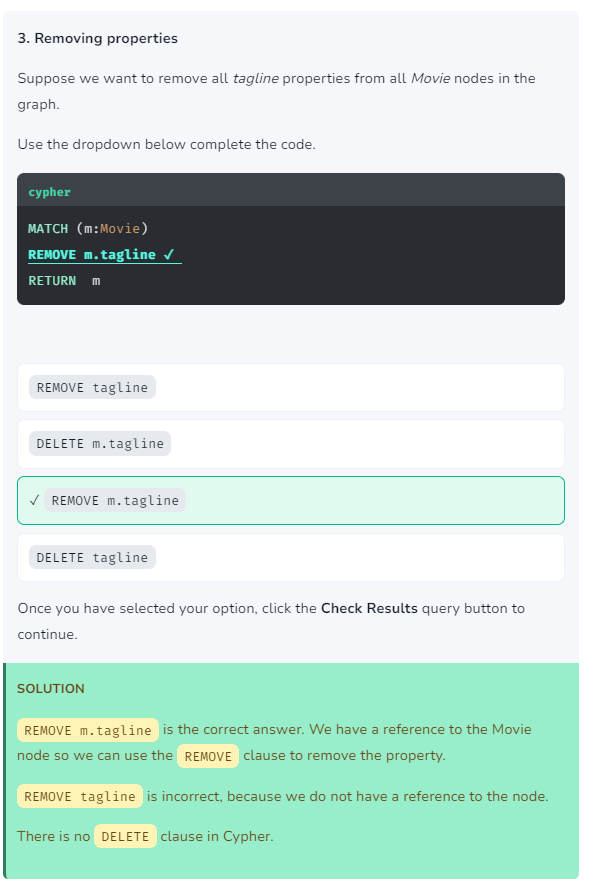


You should never remove the property that is used as the primary key for a node.

### Question(s) / Examples(s)-

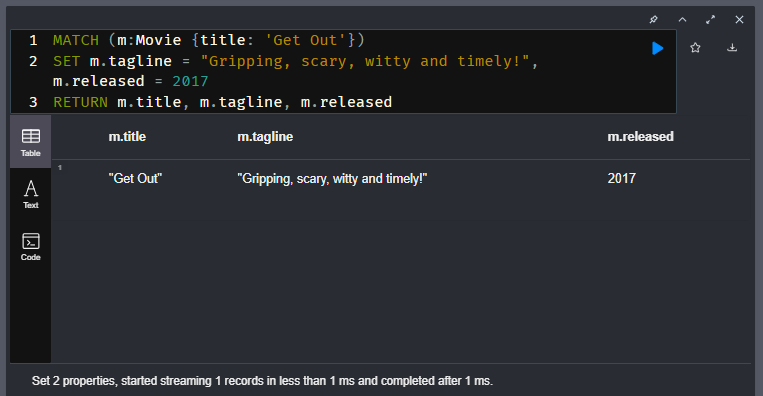






Another Example-

Updating the tagline and release year of the movie



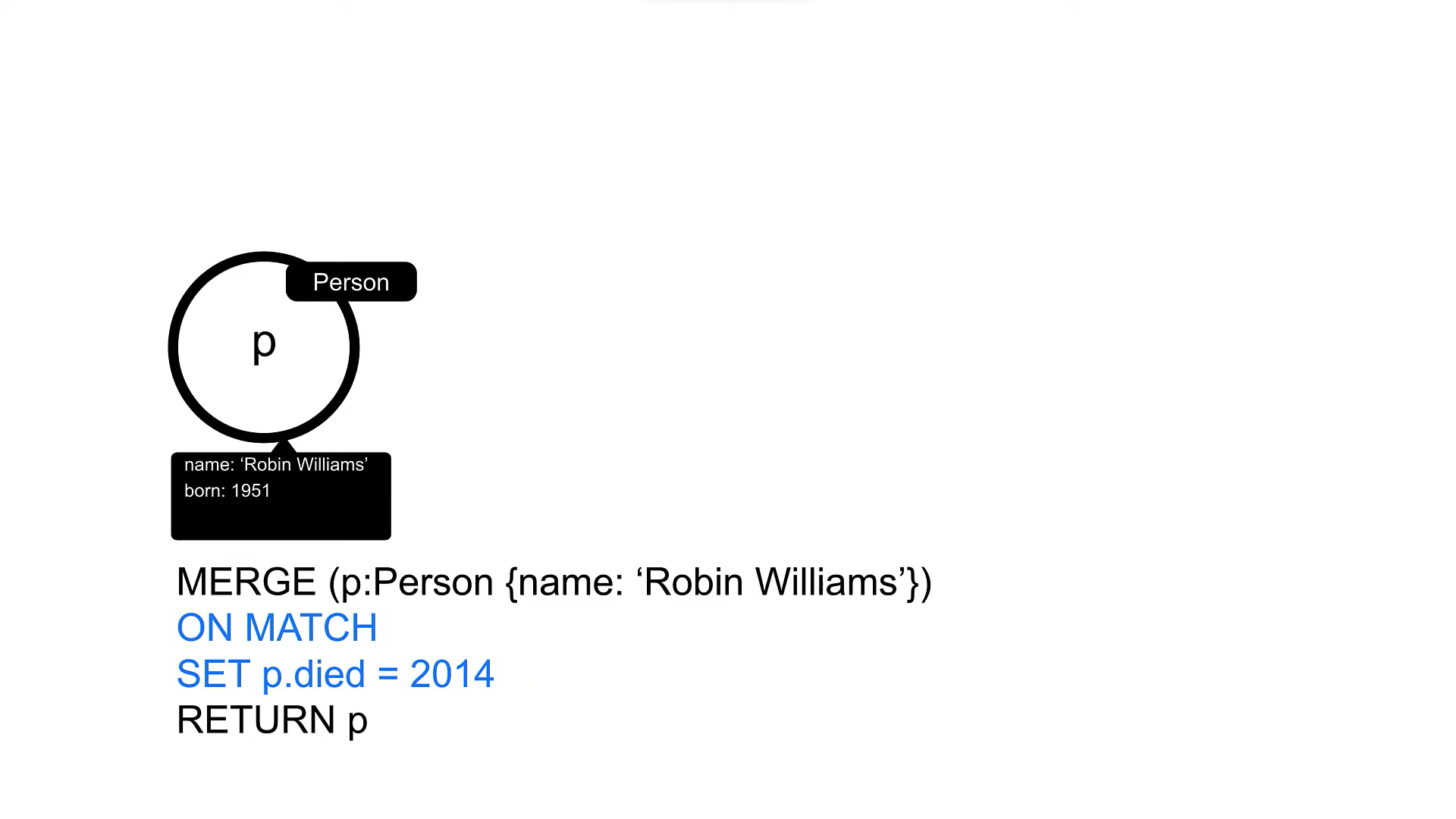
# Merge Processing

Youtube link: <https://youtu.be/3zHH1ei5eCw>

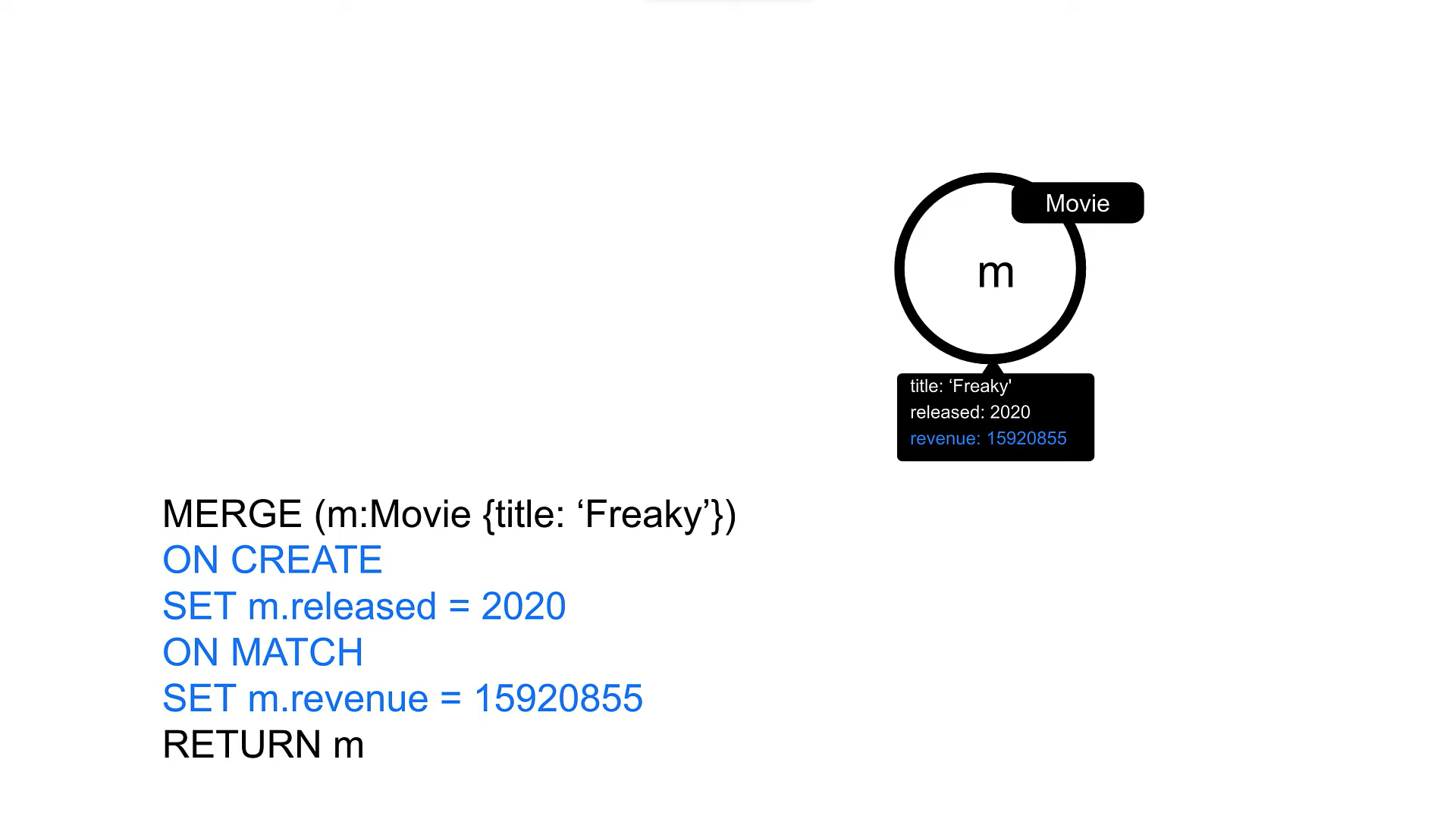
Transcript-

Sandbox neo4j lesson is there and contains screenshots

## Merge processing



Looks for the node and if its found then we update accordingly



You have learned that you can use MERGE to create nodes and relationships in the graph. MERGE operations work by first trying to find a pattern in the graph. If the pattern is found then the data already exists and is not created. If the pattern is not found, then the data can be created.

### Customizing **MERGE** behavior

You can also specify behavior at runtime that enables you to set properties when the node is created or when the node is found. We can use the ON CREATE SET or ON MATCH SET conditions, or the SET keywords to set any additional properties.

In this example, if the Person node for McKenna Grace does not exist, it is created and the createdAt property is set. If the node is found, then the updatedAt property is set. In both cases, the born property is set.

Run this Cypher code at least 2 times to observe what properties are set. You can see the properties in table view.

**cypher**

Copy to ClipboardRun in Sandbox

*// Find or create a person with this name*

MERGE (p:Person {name: 'McKenna Grace'})

*// Only set the `createdAt` property if the node is created during this query*

ON CREATE SET p.createdAt = datetime()

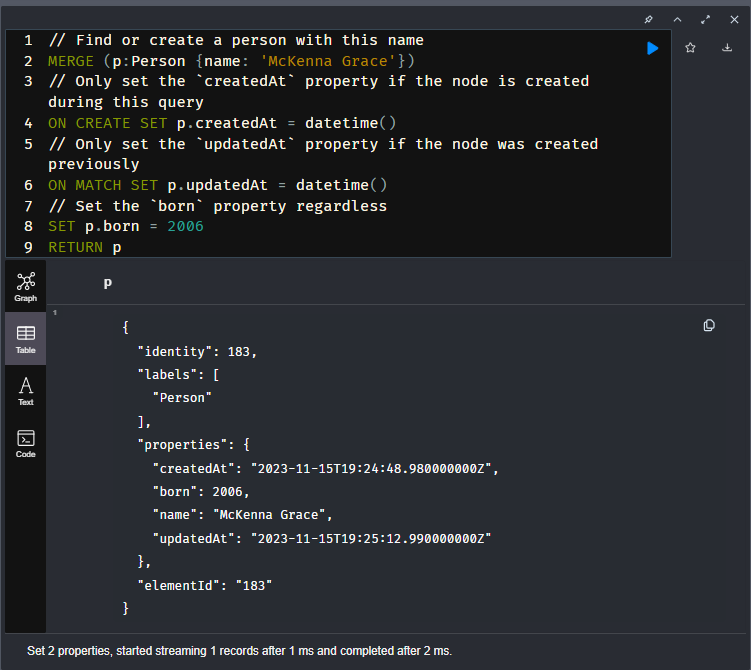
*// Only set the `updatedAt` property if the node was created previously*

ON MATCH SET p.updatedAt = datetime()

*// Set the `born` property regardless*

SET p.born = 2006

RETURN p



If you want to set multiple properties for an ON CREATE SET or ON MATCH SET clause, you separate them by commas. For example:

ON CREATE SET m.released = 2020, m.tagline = `A great ride!'

### Merging with relationships

You can use MERGE to create nodes or relationships:

Run this Cypher code:

**cypher**

Copy to ClipboardRun in Sandbox

*// Find or create a person with this name*

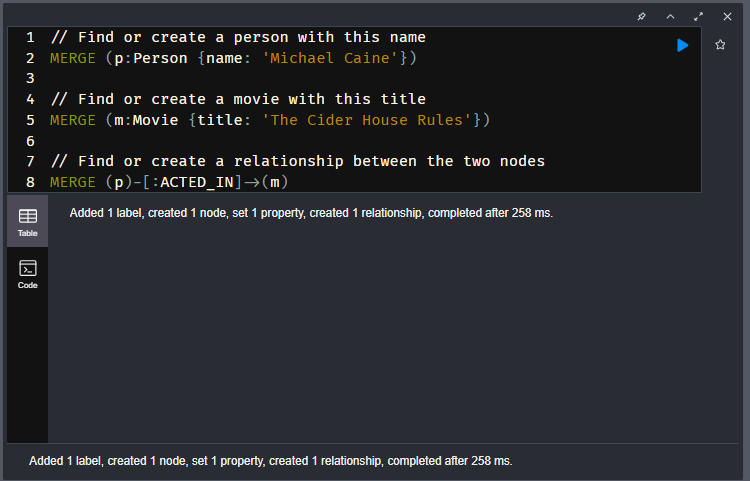
MERGE (p:Person {name: 'Michael Caine'})

*// Find or create a movie with this title*

MERGE (m:Movie {title: 'The Cider House Rules'})

*// Find or create a relationship between the two nodes*

MERGE (p)-[:ACTED\_IN]->(m)



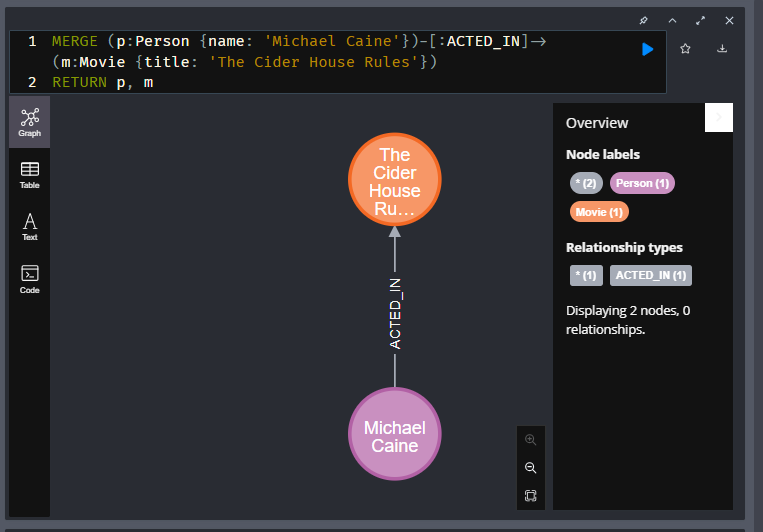
Another way your can create these nodes and relationship is as follows:

**cypher**

Copy to ClipboardRun in Sandbox

MERGE (p:Person {name: 'Michael Caine'})-[:ACTED\_IN]->(m:Movie {title: 'The Cider House Rules'})

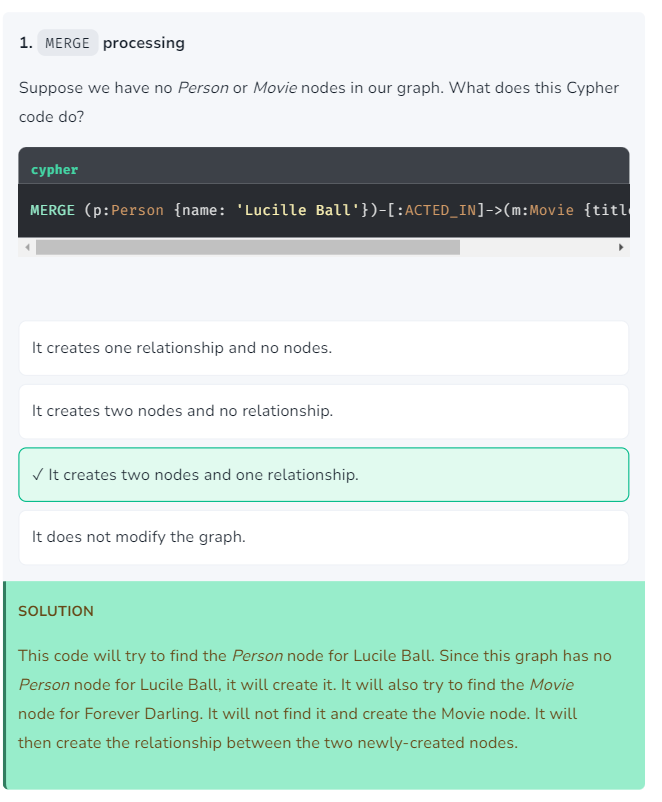
RETURN p, m

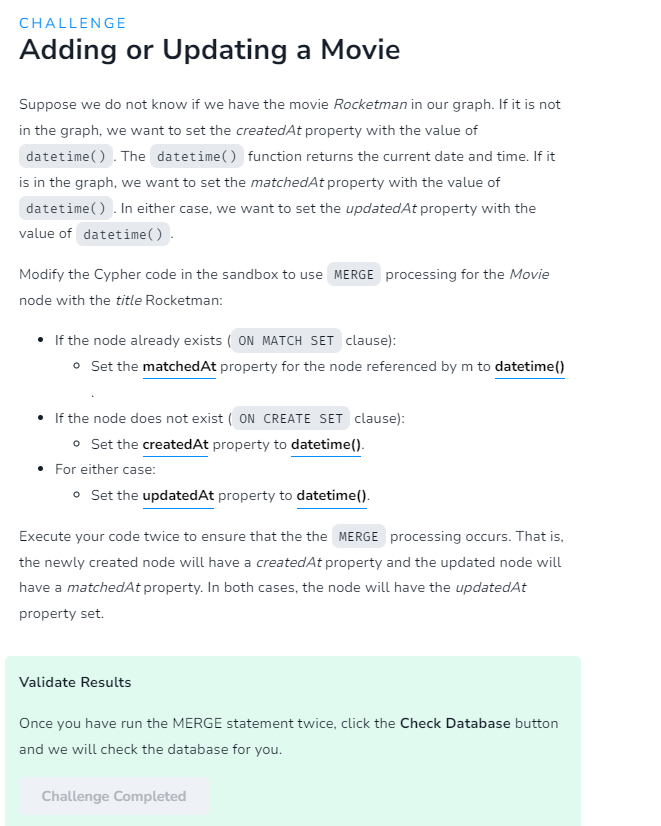


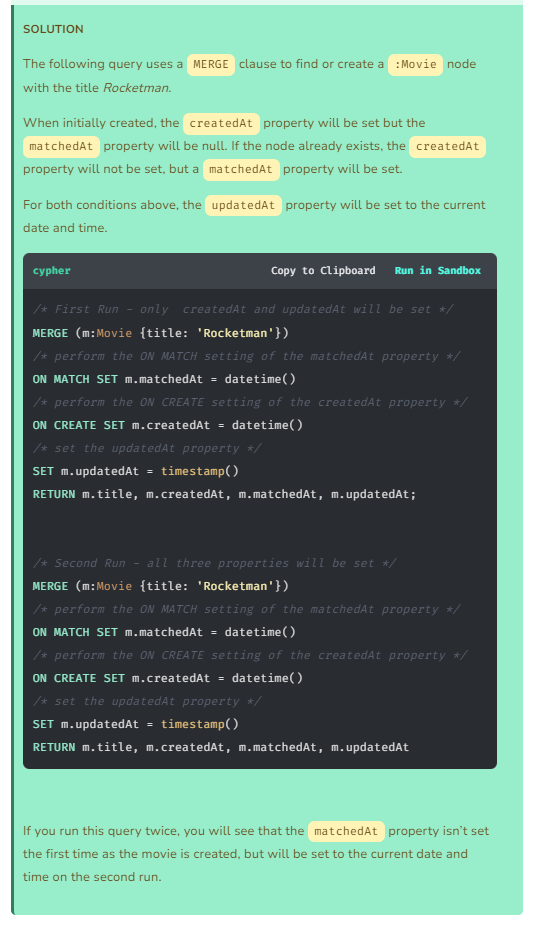
Here is what happens in the query processor:

1. Neo4j will attempt to find a Person node with the name Michael Caine.
2. If it does not exist, it creates the node.
3. Then, it will attempt to expand the ACTED\_IN relationships in the graph for this node.
4. If there are any ACTED\_IN relationships from this node, it looks for a Movie with the title 'The Cider House Rules'.
5. If there is no node for the Movie, it creates the node.
6. If there is no relationship between the two nodes, it then creates the ACTED\_IN relationship between them.

### Question(s) / Example(s)-







# Deleting Data

Youtube link: <https://youtu.be/qNY4ltwe8po>

Transcript-

Sandbox neo4j lesson is there and contains screenshots

## Deleting data

In a Neo4j database you can delete:

* nodes
* relationships
* properties
* labels

To delete any data in the database, you must first retrieve it, then you can delete it. You have already learned how to remove or delete properties from nodes or relationships.

### Deleting a node

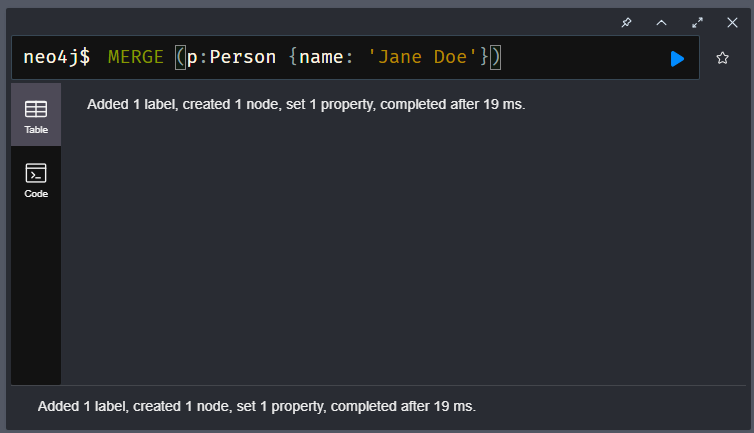
Suppose you have created a Person node for Jane Doe.

Run this Cypher code to create the node:

**cypher**

Copy to ClipboardRun in Sandbox

MERGE (p:Person {name: 'Jane Doe'})



You delete this node as follows where you first retrieve the node. Then with a reference to the node you can delete it.

**cypher**

Copy to ClipboardRun in Sandbox

MATCH (p:Person)

WHERE p.name = 'Jane Doe'

DELETE p



### Deleting a relationship

Suppose we had our Jane Doe node again where she was added as an actor in the movie, The Matrix. Run this code to create the node and the relationship.

**cypher**

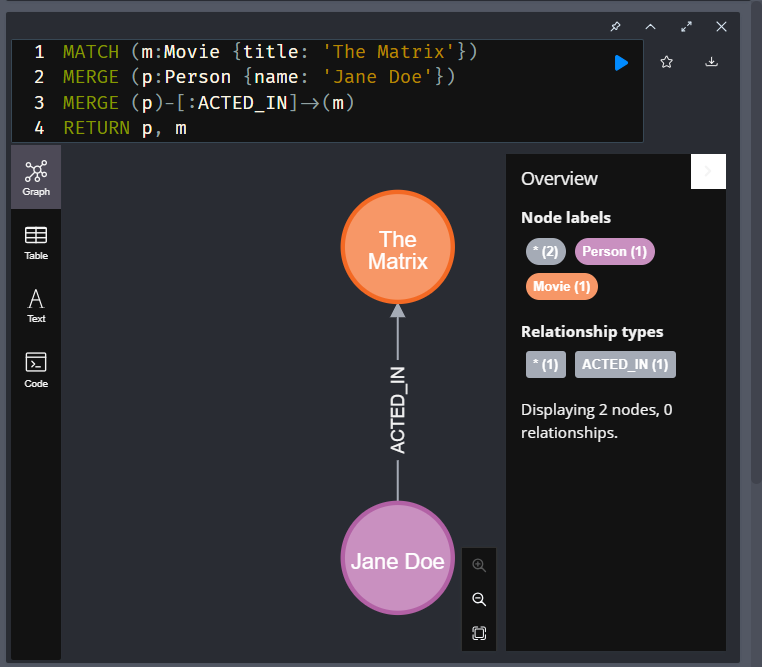
Copy to ClipboardRun in Sandbox

MATCH (m:Movie {title: 'The Matrix'})

MERGE (p:Person {name: 'Jane Doe'})

MERGE (p)-[:ACTED\_IN]->(m)

RETURN p, m



This code creates one node and the relationship from Jane Doe to The Matrix.

To leave the Jane Doe node in the graph, but remove the relationship we retrieve the relationship and delete it.

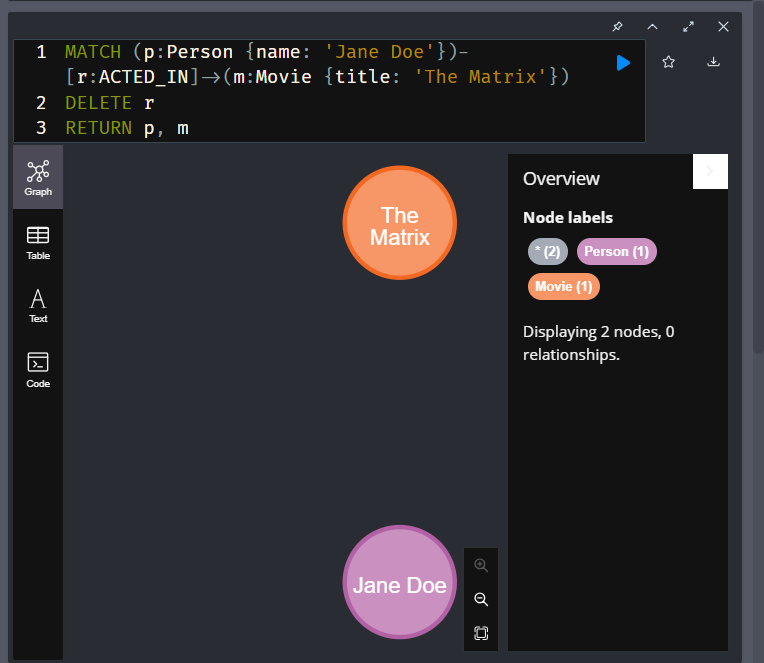
**cypher**

Copy to ClipboardRun in Sandbox

MATCH (p:Person {name: 'Jane Doe'})-[r:ACTED\_IN]->(m:Movie {title: 'The Matrix'})

DELETE r

RETURN p, m



Run this Cypher code that recreates the relationship:

**cypher**

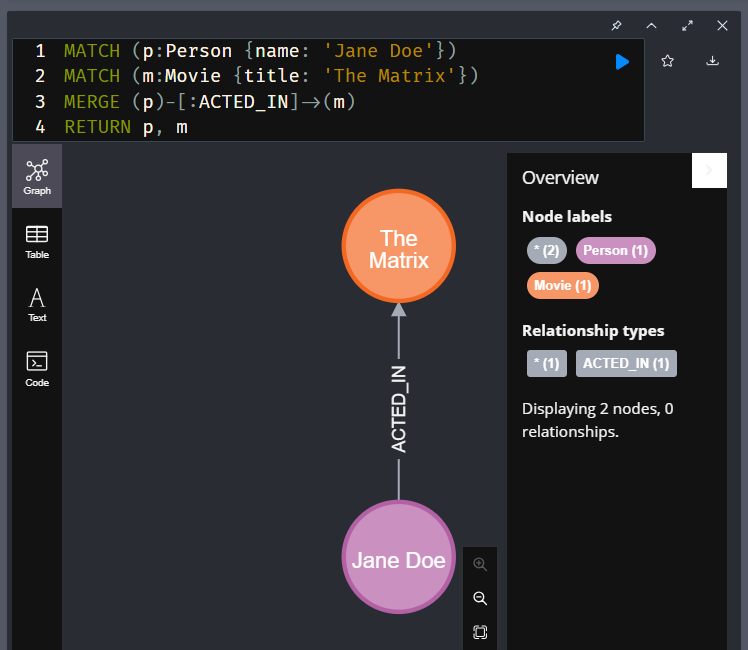
Copy to ClipboardRun in Sandbox

MATCH (p:Person {name: 'Jane Doe'})

MATCH (m:Movie {title: 'The Matrix'})

MERGE (p)-[:ACTED\_IN]->(m)

RETURN p, m



If we attempt to delete the Jane Doe node, we will receive an error because it has relationships in the graph.

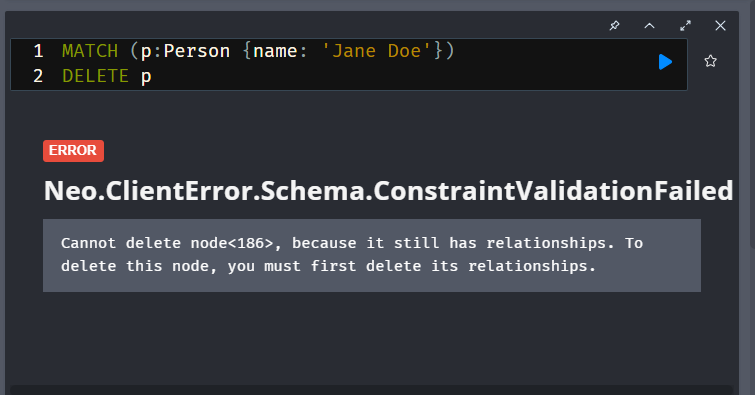
Try running this Cypher code:

**cypher**

Copy to ClipboardRun in Sandbox

MATCH (p:Person {name: 'Jane Doe'})

DELETE p



You should receive an error. Neo4j prevents orphaned relationships in the graph.

### Deleting a node and its relationships

Neo4j provides a feature where you cannot delete a node if it has incoming or outgoing relationships. This prevents the graph from having orphaned relationships.

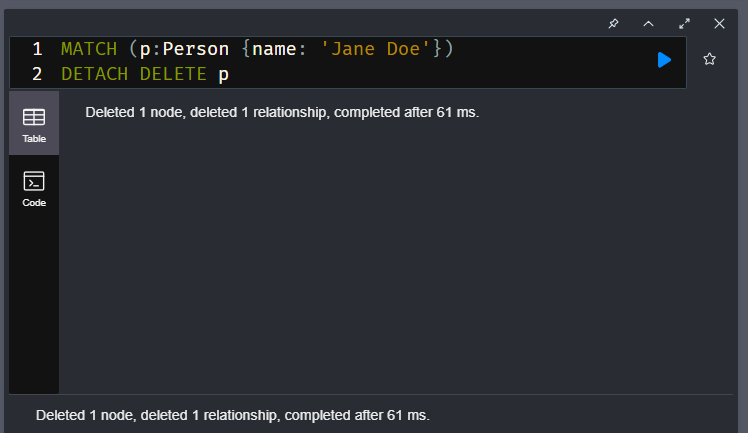
Run this Cypher code:

**cypher**

Copy to ClipboardRun in Sandbox

MATCH (p:Person {name: 'Jane Doe'})

DETACH DELETE p



This code deletes the relationship and the Person node.

You can also delete all nodes and relationships in a database with this code.

Do NOT delete all nodes and relationships in the graph as you need them for the next challenge!

**cypher**

MATCH (n)

DETACH DELETE n

You should only do this on relatively small databases as trying to do this on a large database will exhaust memory.

### Deleting labels

A best practice is to have at least one label for a node, but not more than four.

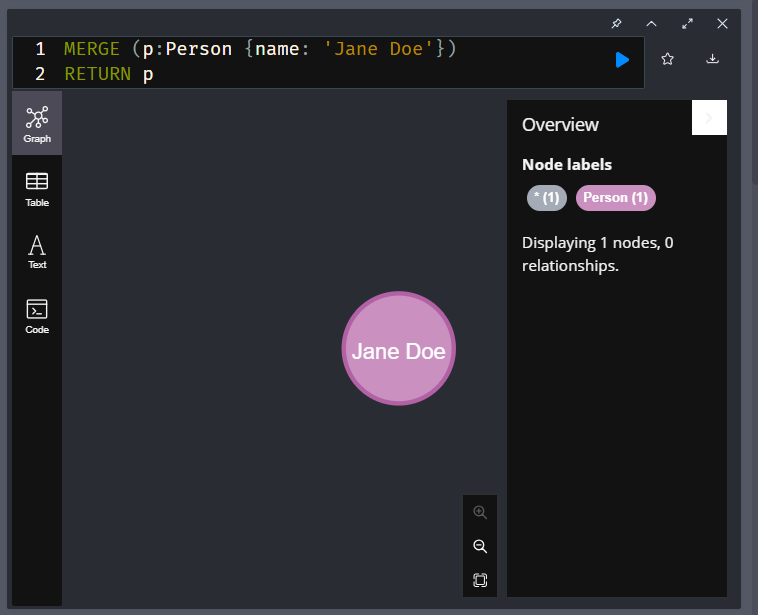
Run this Cypher code to create this Person node in the graph:

**cypher**

Copy to ClipboardRun in Sandbox

MERGE (p:Person {name: 'Jane Doe'})

RETURN p



Next, run this code to add a new label to this node:

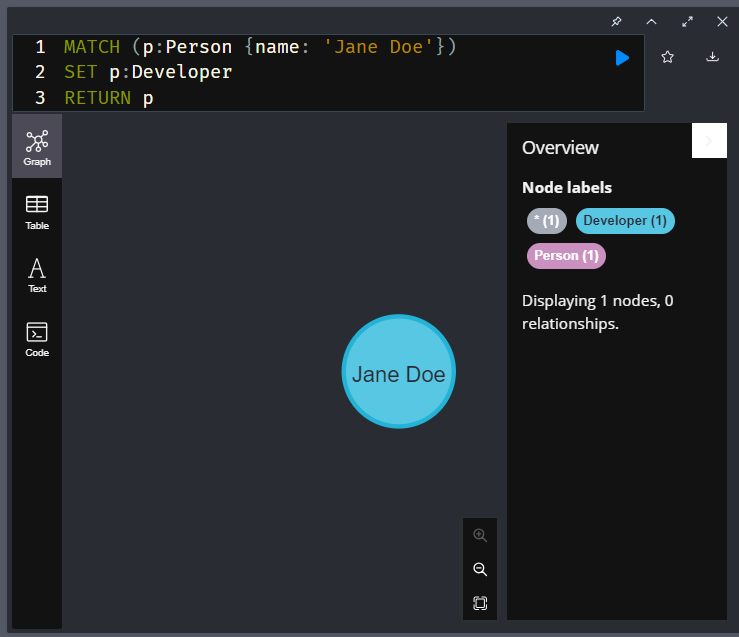
**cypher**

Copy to ClipboardRun in Sandbox

MATCH (p:Person {name: 'Jane Doe'})

SET p:Developer

RETURN p



To remove the newly-added label, Developer, you use the REMOVE clause. Run this code:

**cypher**

Copy to ClipboardRun in Sandbox

MATCH (p:Person {name: 'Jane Doe'})

REMOVE p:Developer

RETURN p



The Jane Doe node has two labels, Person and Developer. You can use a MATCH to find that node. Note that you could have specified MATCH (p:Developer {name: 'Jane Doe'}) or MATCH (p:Person:Developer {name: 'Jane Doe'}) to find the same node. Once we have a reference to that node, we can remove the label with the REMOVE clause.

And finally, delete the Jane Doe node by running this code:

**cypher**

Copy to ClipboardRun in Sandbox

MATCH (p:Person {name: 'Jane Doe'})

DETACH DELETE p



Note that DELETE p would also work in this case since we have not created any relationships.

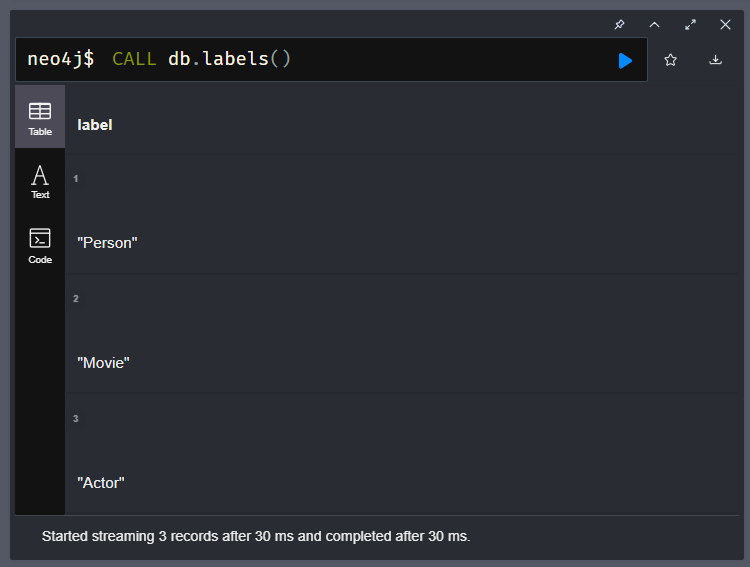
### What labels exist in the graph?

This code returns all node labels defined in the graph.

**cypher**

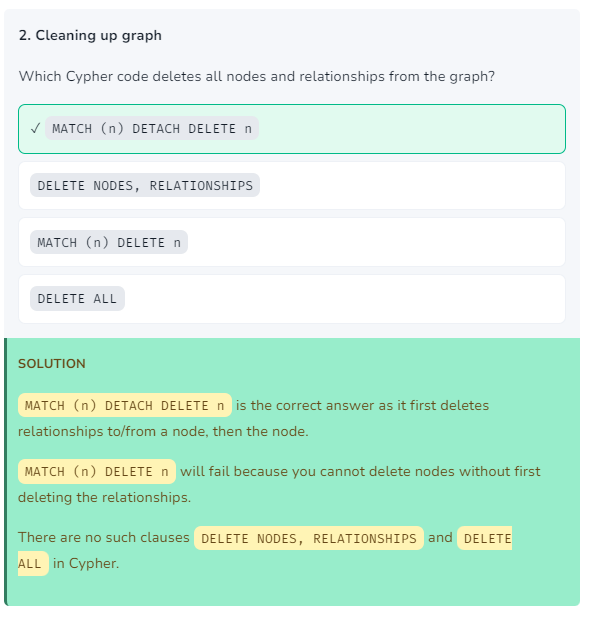
Copy to ClipboardRun in Sandbox

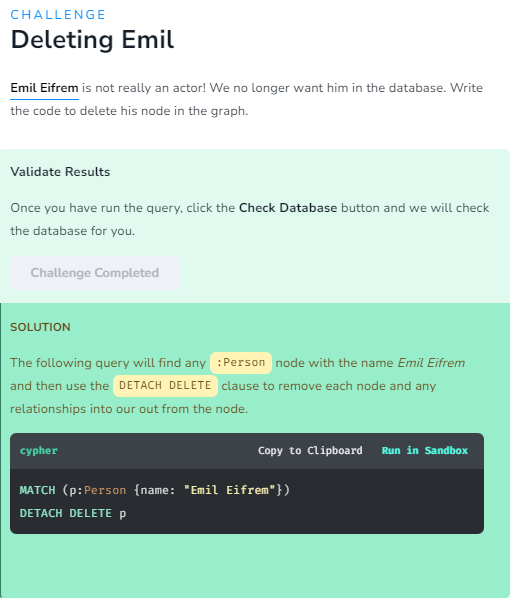
CALL db.labels()



### Questions / Examples-







# Course Summary

In this course, you have learned how to:

* Write Cypher code to query the database.
* Write Cypher code to modify the database.
* Use some Cypher statement best practices.

## Resources

* **There are many resources available to you for learning more about Neo4j**  
  [**https://neo4j.com/developer/resources/**](https://neo4j.com/developer/resources/)
* **Neo4j Community Site where you can ask or answer questions about Neo4j and discuss with other users:**  
  [**https://community.neo4j.com**](https://community.neo4j.com/)
* **Neo4j documentation:**  
  [**https://neo4j.com/docs/**](https://neo4j.com/docs/)
* **Neo4j Sandboxes for experimenting with graphs:**  
  [**https://sandbox.neo4j.com/?ref=graph-academy**](https://sandbox.neo4j.com/?ref=graph-academy)
* **Videos on the Neo4j YouTube channel:**  
  [**https://www.youtube.com/channel/UCvze3hU6OZBkB1vkhH2lH9Q**](https://www.youtube.com/channel/UCvze3hU6OZBkB1vkhH2lH9Q)
* **Become a Neo4j certified developer:**  
  [**https://graphacademy.neo4j.com/categories/certification/**](https://graphacademy.neo4j.com/categories/certification/)
* **GitHub repository:**  
  [**https://github.com/neo4j-contrib**](https://github.com/neo4j-contrib)
* **Neo4j events all over the world:**  
  [**https://neo4j.com/events/world/all/**](https://neo4j.com/events/world/all/)
* **Graph Gists for learning more use cases for Neo4j:**  
  [**https://neo4j.com/graphgists/**](https://neo4j.com/graphgists/)
* **Attend a Neo4j meetup:**  
  [**https://www.meetup.com/topics/neo4j/**](https://www.meetup.com/topics/neo4j/)
* **View questions/answers raised about Neo4j:**  
  [**https://stackoverflow.com/tags/neo4j/hot**](https://stackoverflow.com/tags/neo4j/hot)