Creating Nodes and Relationships

Table of Contents

About this module

You have learned how to query a graph using a number of Cypher clauses and keywords that start with the `MATCH` clause. You learned how to retrieve data based upon label values, property key values, and relationship types, and how to perform some useful intermediate processing during a query to control what data is returned.

In this module, you will learn how to add nodes and relationships to a graph, and how to update existing data in a graph. First you will learn how to create nodes in the graph, as well as add and remove labels and properties for nodes. Then you will learn how to create relationships in the graph, as well as add and remove properties for a relationship. Next, you will lear how to delete nodes and relationships in the graph. Finally, you will learn how to merge data in a graph.

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

- Create a node
  - Add and remove node labels
  - Add and remove node properties
  - Update properties
- Create a relationship
  - Add and remove properties for a relationship
- Delete a node
- Delete a relationship
- Merge data in a graph
  - Creating nodes
  - Creating relationships

Creating nodes

Recall that a node is an element of a graph that has zero or more labels, properties, and relationships to or from another node in the graph.

When you create a node, you can add it to the graph without connecting it to another node.

Here is the simplified syntax for creating a node

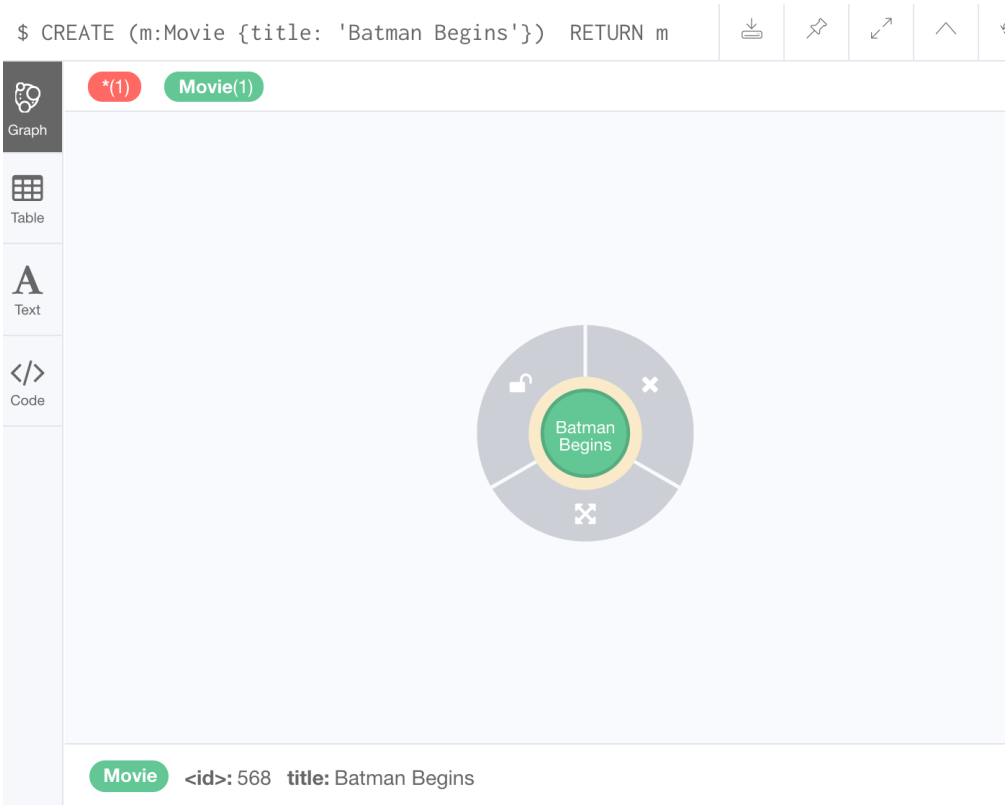
```
CREATE (`optional-variable` `optional-labels` `{optional-properties}`)
```

If you plan on referencing the newly created node, you must provide a variable. Whether you provide labels or properties at node creation time is optional. In most cases, you will want to provide some label and property values for a node when created. This will enable you to retrieve the node. Provided you have a reference to the node (e.g. using a `MATCH` clause), you can always add labels and properties at a later time.

Here are some examples of creating a single node in Cypher:

`CREATE (:Movie {title: 'Batman Begins'})`	Adds a node to the graph of type *Movie* with the *title Batman Begins*. This node can be retrieved using the title.
`CREATE (:Movie:Action {title: 'Batman Begins'})`	Adds a node to the graph of types_Movie_ and *Action* with the *title Batman Begins*. This node can be retrieved using the title.
`CREATE (m:Movie:Action {title: 'Batman Begins'})`	Adds a node to the graph of types_Movie_ and *Action* with the *title Batman Begins*. This node can be retrieved using the title. The variable *m* can be used for later processing after the `CREATE` clause.
`CREATE (m:Movie:Action {title: 'Batman Begins'}) RETURN m.title`	Adds a node to the graph of types_Movie_ and *Action* with the *title Batman Begins*. This node can be retrieved using the title. Here we return the title of the node.

Here is what you see in Neo4j Browser when you create a node for the movie, *Batman Begins* where we have selected the node returned to view its properties.:

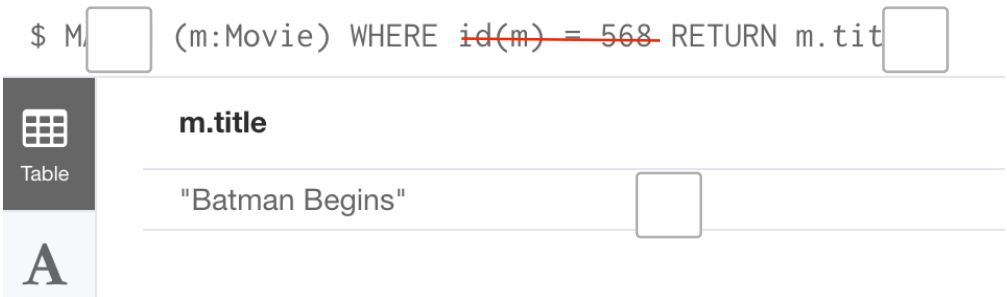


When the graph engine creates a node, it automatically assigns a read-only, u e ID to the node. Here we see that the *id* of the node is *568*. Cypher has a built-in function `id()` that returns the value of *id* for a node. In your application, you may want t erence key nodes by their *id* value for fast retrieval.

For example, this Cypher statement will retrieve the node we just created:

```
MATCH (m:Movie)
WHERE id(m) = 568
RETURN m.title
```

Here is what you see in Neo4j Browser when you retrieve the nod ts *id* value:



After you have created a node, you can add more p rties or labels to it and most importantly, connect it to another node.

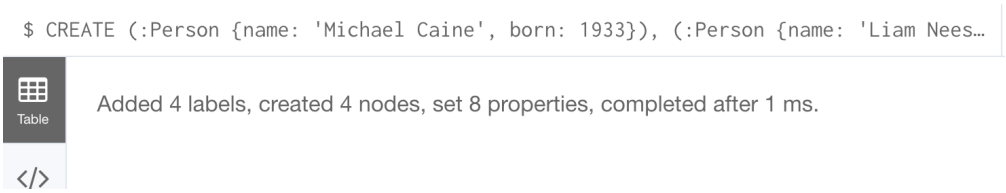
Creating multiple nodes

You can create multiple nodes by simply separating the nodes specified with comm

Here is an example, where we create some *Person* nodes that will represent some of the people associated with the movie *Batman Begins*.

```
CRE :Person {name: 'Michael Caine', born: 1933}),
(:Person {name: 'Liam Neeson', born: 1952}),
(:Person {name: 'Katie Holmes', born: 1978}),
(:Person {name: 'Benjamin Melniker', born: 1913})
```

Here is the result of running this Cypher code:



Important: The gr ngine will create a node with the same properties of a node that already exists. You can prevent this from happening in one of two ways:

1. You can use `MERGE` rather than `CREATE` when creating the node.
2. You can add constraints to your graph.

You will learn about merging data later in this module. Constraints are configur r a graph and are covered later in this training.

Adding labels to a node

You may not know ahead of time what label or labels you want for a node when it is created. You can add labels to a node using the `SET` clause.

Here is the simplified syntax for adding labels to a node

```
// `x` i  e variable reference to the node
SET `x`  el`                 // adding one label
|
// `x` is the variable reference to the node
SET `x`  el1:label2`         // adding two labels
```

If you attempt to add a label to a node where abel already exists, the `SET` processing is ignored.

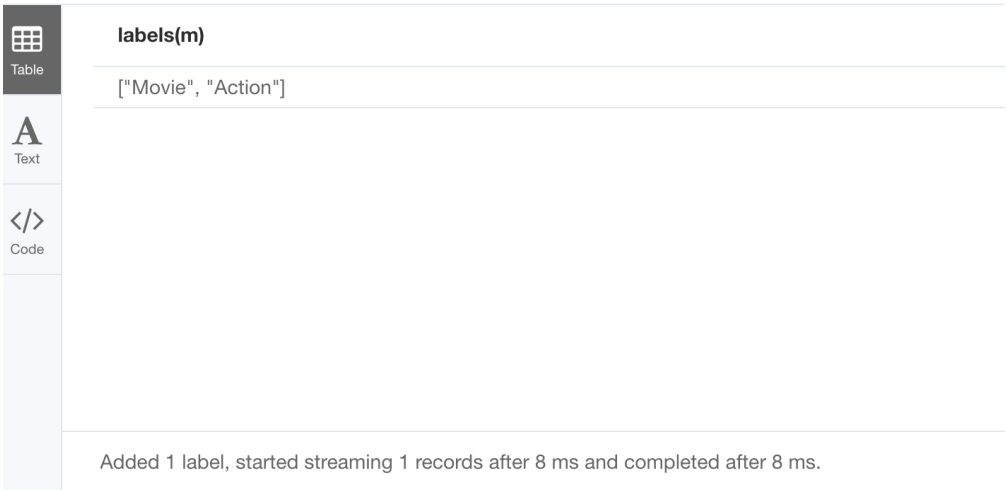
Here is an example where we add the *Action* label to the node that has a label, *Movie*:

```
MATCH (m:Movie)
WHERE m.title = 'Batman Begins'
SET m:Action
RETURN labels(m)
```

Assuming that we have previously created the node for the movie, here is the result of running this

Cypher c

```
$ MATCH (m:Movie) WHERE m.title = 'Batman Begins' SET m:Action RETURN labels(m)
```



Notice here that we call the built-in function, `labels()` that returns the set of labels for the node.

Removing labels from a node

Perhaps your data model has changed or the underlying data for a node has changed so that the label for a node is no longer useful or valid.

Here is the simplified syntax for removing a label from a node:

```
// `x` is the variable reference to the node
REMOVE x:`label`
```

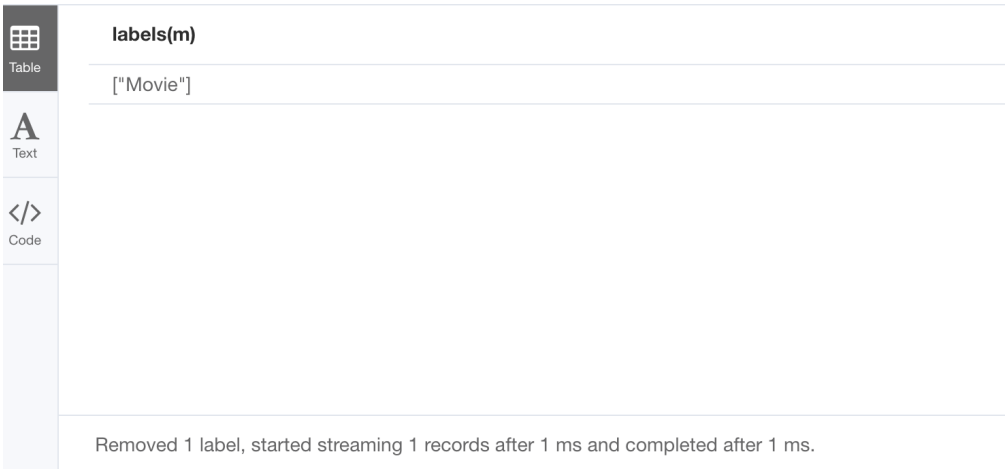
If you attempt to remove a label from a node where label does not exist, the `SET` processing is ignored.

Here is an example where we remove the *Action* label from the node that has a labels, *Movie* and *Action*:

```
MATCH (m:Movie)
WHERE m.title = 'Batman Begins'
REMOVE m:Action
RETURN labels(m)
```

Assuming that we have previously created the node for the movie, here is the result of running this Cypher code:

```
$ MATCH (m:Movie) WHERE m.title = 'Batman Begins' REMOVE m:Action RETURN labels(m)
```



Adding properties to a node

After you have created a node and have a reference to the node, you can add properties to the node, again using the SET keyword.

Here is the simplified syntax for adding properties to a node

```
// `x` is the variable reference to the node
SET `x`.`property-name` = `value`
|
SET `x`.`property-name1` = `value1` , `x`.`property-name2` = `value2`
|
SET `x` = {`property-name1`: `value1`, `property-name2`: `value2`}
```

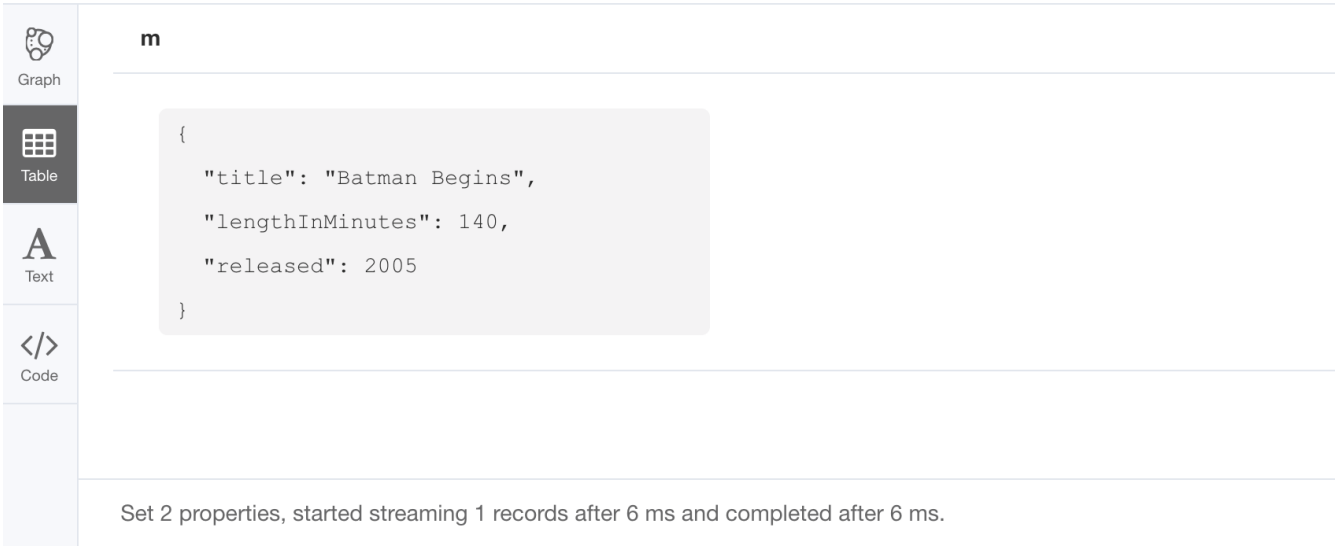
If the property does not exist, it is added to the node. If the property exists, its value is updated. Note that the type of data for a property is not enforced. That is, you can assign a string value to a property that was once a numeric value and visa versa. When specify the JSON-style object for assignment to the node, the object must include all of the properties for the node.

Here is an example where we add the properties *released* and *lengthInMinutes* to the movie *Batman Begins*:

```
MATCH (m:Movie)
WHERE m.title = 'Batman Begins'
SET m.released = 2005, m.lengthInMinutes = 140
RETURN m
```

Assuming that we have previously created the node for the movie, here is the result of running this Cypher code:

```
$ MATCH (m:Movie) WHERE m.title = 'Batman Begins' SET m.released = 2005, m.lengthInMinutes = 140 .
```



Here is another example where we set the properties to the movie node using the JSON-style object containing the property keys and values. Note that all properties must be included in the object.

```
MATCH (m:Movie)
WHERE m.title = 'Batman Begins'
SET  m = {title: 'Batman Begins',
          released: 2005,
          lengthInMinutes: 140,
          videoFormat: 'DVD',
          grossMillions: 206.5}
RETURN m
```

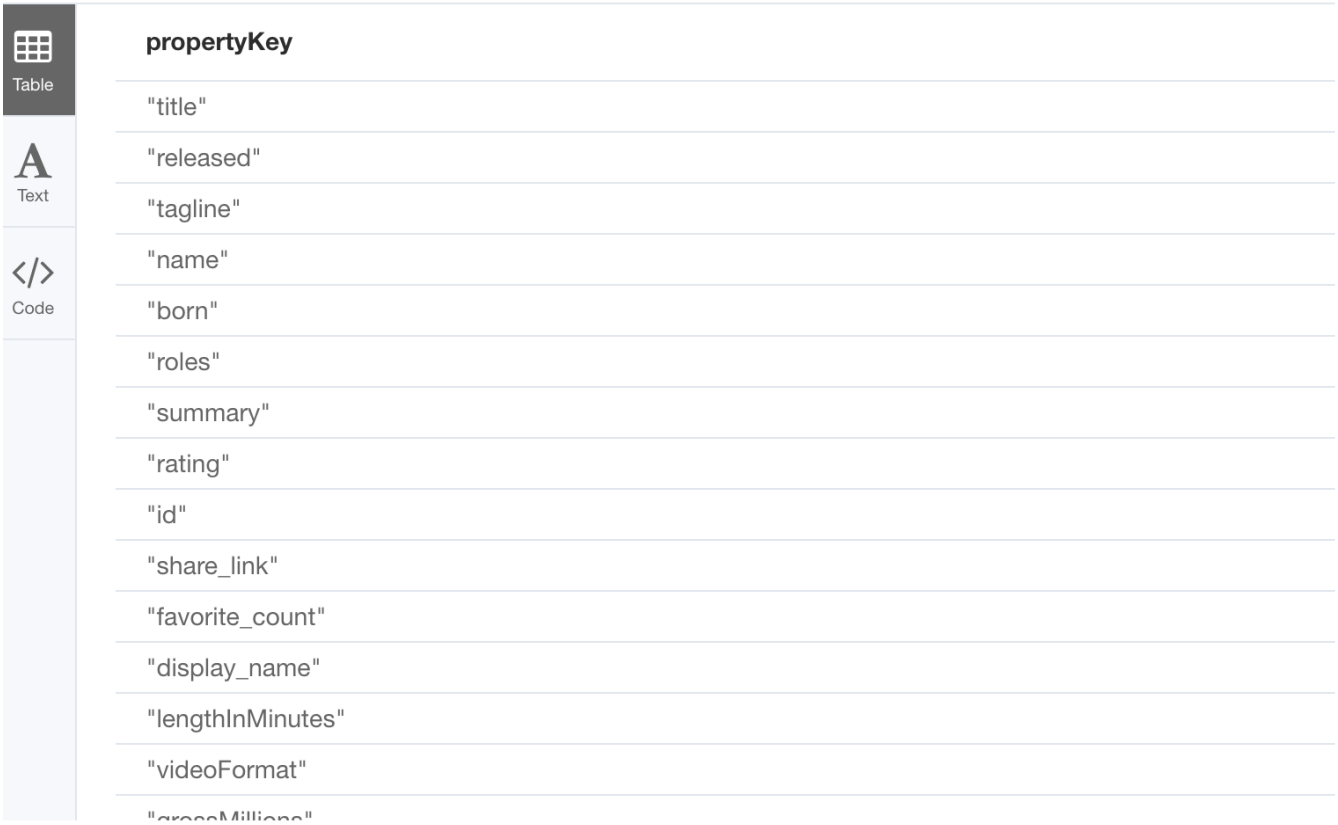
Here is the result of running this Cypher code:

```
$ MATCH (m:Movie) WHERE m.title = 'Batman Begins' SET m = {title: 'Batman Begins',  released: 200…
```

Note that when you add a property to a node for the first time in the graph, the property key is added to the graph. So for example, in the previous example, we added the *videoFormat* and *grossMillions* property keys to the graph as they have never been used before for a node in the

graph. Once a property key is added to the graph, it is never removed. When you examine the property keys in the database (by calling `call db.propertyKeys()`, you will see all property keys created for the graph, regardless of whether they are currently used for nodes and relationships.

```
$ call db.propertyKeys()
```



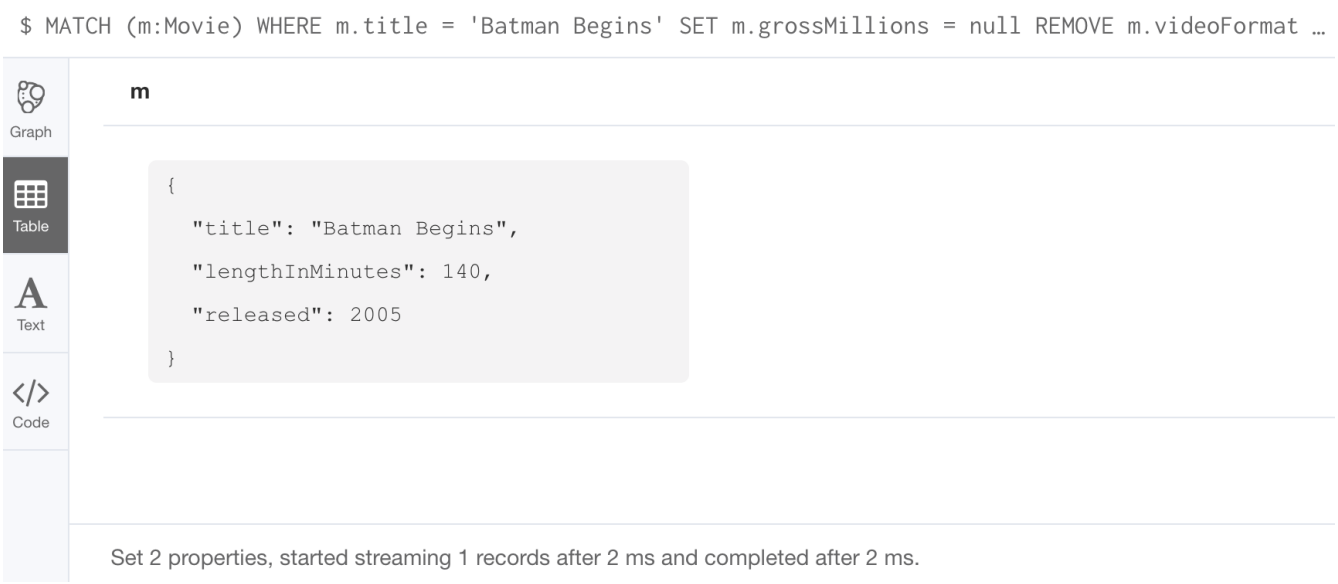
Removing properties from a no

There are two ways that you can remove a property from a node. One way is to use the REMOVE keyword. The other way is to set the property's value to `null`.

Suppose we determined that no other *Movie* node in the graph has the properties, *videoFormat* and *grossMillions*. There is no restriction that nodes of the same type must have the same properties. However, we have decided that we want to remove these properties from this node. Here is the Cypher code to remote this property from this *Batman Begins* node:

```
MATCH (m:Movie)
WHERE m.title = 'Batman Begins'
SET m.grossMillions = null
REMOVE m.videoFormat
RETURN m
```

Assuming that we have previously created the node for the movie with the these properties, here is the result of running this Cypher code where we remove each prop a different way. One way using the `SET` clause to set the property to null. And the other way using the `REMOVE` clause.



Exercise 8: Creating Nodes

In the query edit pane of Neo4j Browser, execute the browser command: `:play http://gu.neo4j.com/intro-neo4j-exercises` and follow the instructions for Exercise 8.

Creating relationships

As you have learned in the previous exercises where you query the graph, very useful types of information in a graph are the connections between nodes, as these connections are typically used to quickly retrieve the data of interest.

Here is the syntax for creating a relationship between two nodes:

```
// `x`  and `y` are the variable reference to the nodes that have been retrieved
CREATE (x)-[:`REL_TYPE`]->(y)
|
CREATE (x)<-[:`REL_TYPE`]-(y)
```

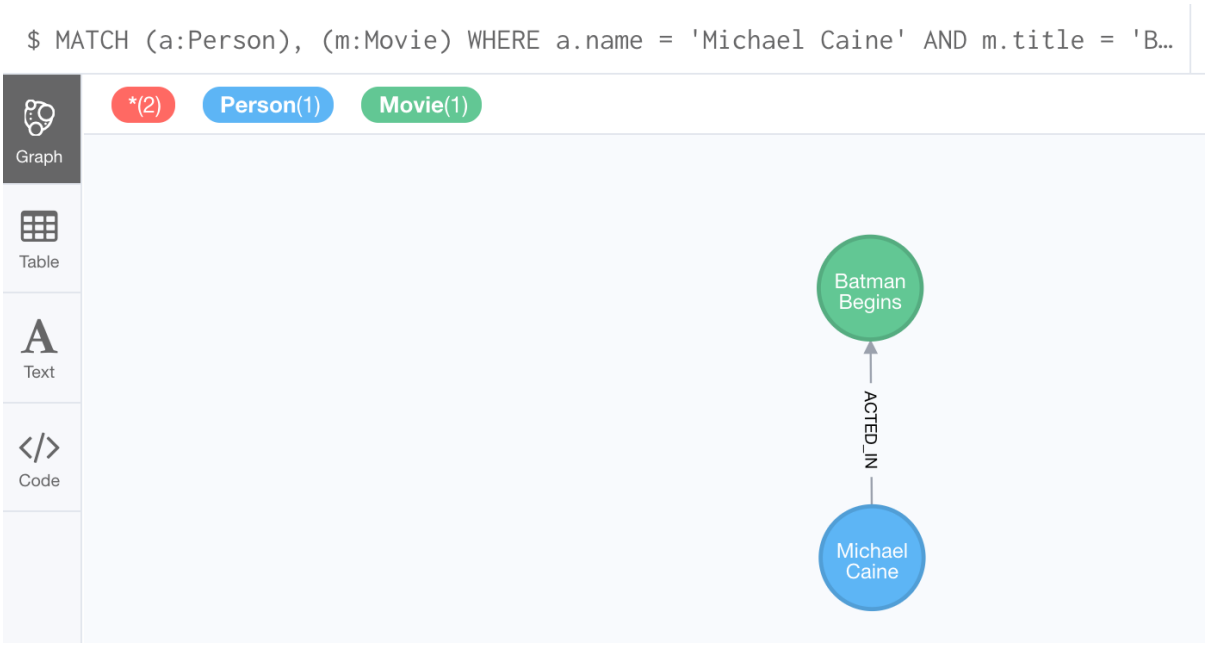
When you create the relationship, it must have direction. You can query nodes for a relationship in either direction, but you must create the relationship with a direction.

In most cases, unless you are connecting nodes at creation time, you will retrieve the two nodes, each with their own variables, specify a `WHERE` clause to find them, and then use the variables to connect them.

Here is an example. We want to connect the actor, *Michael Caine* with the movie, *Batman Begins*. We first retrieve the nodes of interest, then we create the relationship.

```
MATCH (a:Person), (m:Movie)
WHERE a.name = 'Michael Caine' AND m.title = 'Batman Begins'
CREATE (a)-[:ACTED_IN]->(m)
RETURN a, m
```

Here is the result of running this Cypher code:



Before you run this code, you will see a warning in Neo4j Browser that you are creating a query

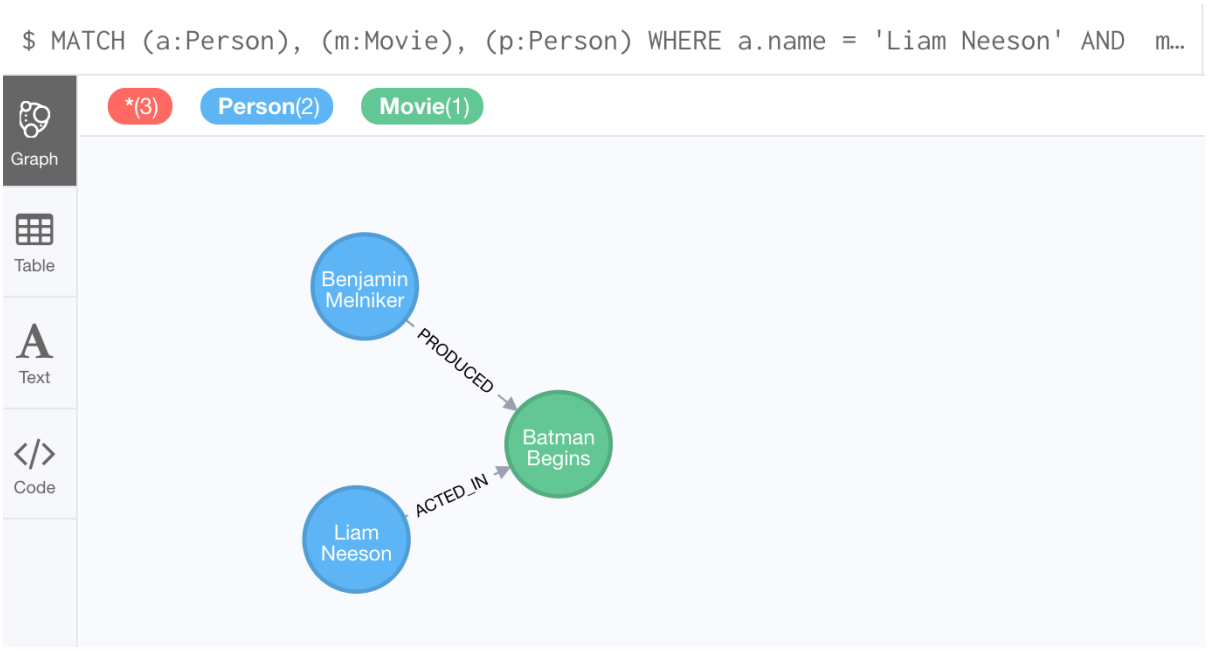
that is a cartesian product that could potentially be a performance issue. If you are familiar with the data in the graph and can be sure that the `MATCH` clauses will not retrieve large amounts of data, you can continue. In our case, we are simply looking up a particular *Person* node and a particular *Movie* node so we can create the relationship.

You can create multiple relationships at once by simply providing the pattern for the creation that includes the relationship types, their directions, and the nodes that you want to connect.

Here is an example where we have already created *Person* nodes for an actor, *Liam Neeson*, and a producer, *Benjamin Melniker*. We create two relationships in this code, one for *ACTED_IN* and one for *PRODUCED*.

```
MATCH (a:Person), (m:Movie), (p:Person)
WHERE a.name = 'Liam Neeson' AND
      m.title = 'Batman Begins' AND
      p.name = 'Benjamin Melniker'
CREATE (a)-[:ACTED_IN]->(m)<-[:PRODUCED]-(p)
RETURN a, m, p
```

Here is the result of running this Cypher code:



Adding properties to relationships

You can add properties to a relationship, just as you add properties to a node. You use the `SET` clause to do so.

Here is the simplified syntax for adding properties to a relationship

```
// `r` is the variable reference to the relationship
SET `r`.`property-name` = `value`
|
SET `r`.`property-name1` = `value1` , `x`.`property-name2` = `value2`
|
SET `r` = {`property-name1`: `value1`, `property-name2`: `value2`}
```

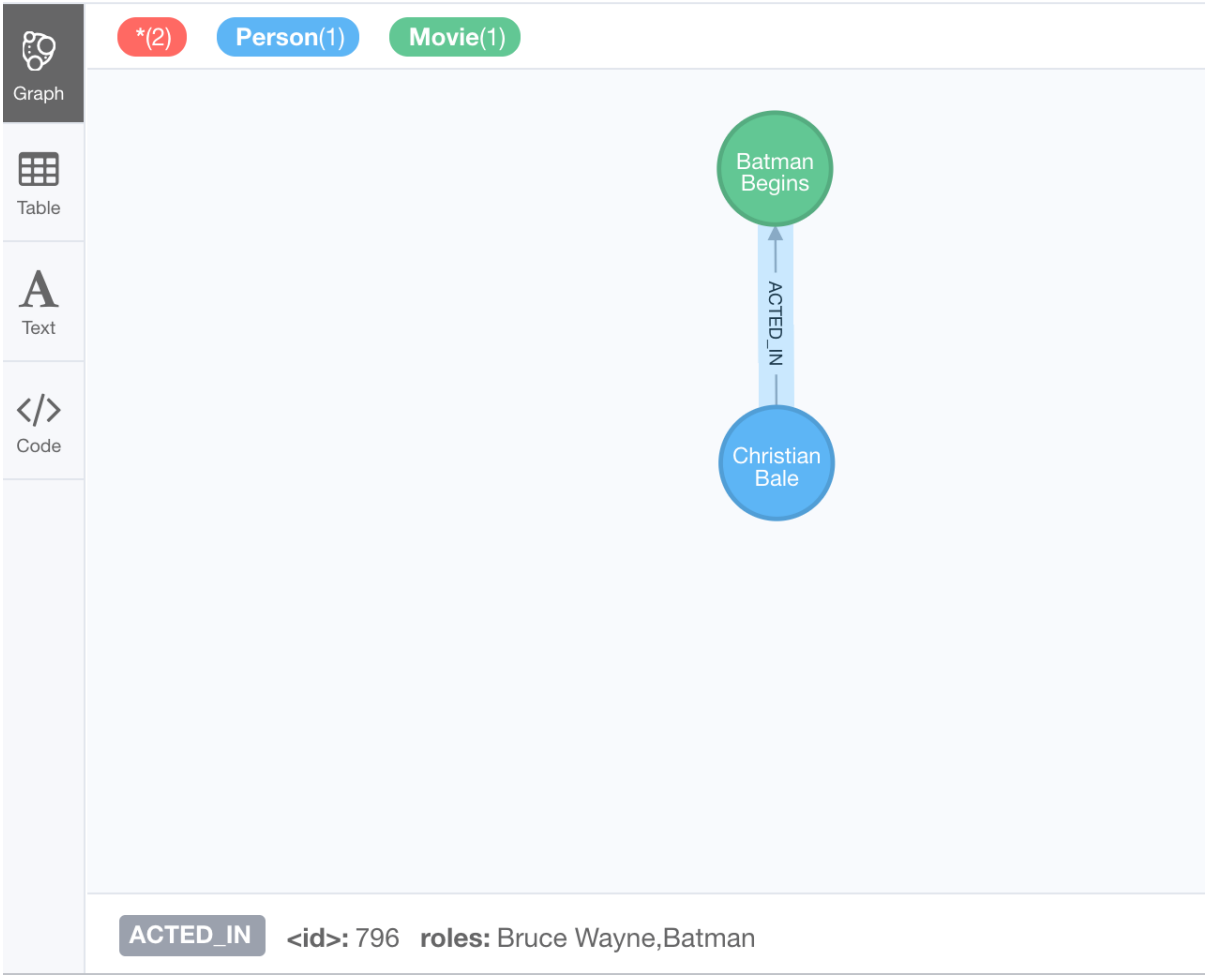
If the property does not exist, it is added to the relationship. If the property exists, its value is updated for the relationship. When specify the JSON-style object for assignment to the relationship, the object must include all of the properties for the relationship, just as you need to do for nodes.

Here is an example where we will add the *roles* property to the *ACTED_IN* relationship from *Christian Bale* to *Batman Begins* right after we have created the relationship.

```
MATCH (a:Person), (m:Movie)
WHERE a.name = 'Christian Bale' AND m.title = 'Batman Begins'
CREATE (a)-[rel:ACTED_IN]->(m)
SET rel.roles = ['Bruce Wayne','Batman']
RETURN a, m
```

Here is the result of running this Cypher code:

```
$ MATCH (a:Person), (m:Movie) WHERE a.name = 'Christian Bale' AND m.title
```



The *roles* property is a list so we add it as such. If the relationship had multiple properties, we could have added them as a comma separated list or as an object, like can do for node properties.

You can also add properties to a relationship when the relationship is created. Here is another way to create and add the properties for the relationship:

```
MATCH (a:Person), (m:Movie)
WHERE a.name = 'Christian Bale' AND m.title = 'Batman Begins'
CREATE (a)-[:ACTED_IN {roles: ['Bruce Wayne', 'Batman']}]->(m)
RETURN a, m
```

By default, the graph engine will create a relationship between two nodes, even if one already exists. You can test to see if the relationship exists before you create it as follows:

```
MATCH (a:Person),(m:Movie)
WHERE a.name = 'Christian Bale' AND
      m.title = 'Batman Begins' AND
      NOT exists((a)-[:ACTED_IN]->(m))
CREATE (a)-[rel:ACTED_IN]->(m)
SET rel.roles = ['Bruce Wayne','Batman']
RETURN a, rel, m
```

Note: You can prevent duplication of relationships by merging data using the `MERGE` clause, rather than the `CREATE` clause. You will learn about merging data later in this module.

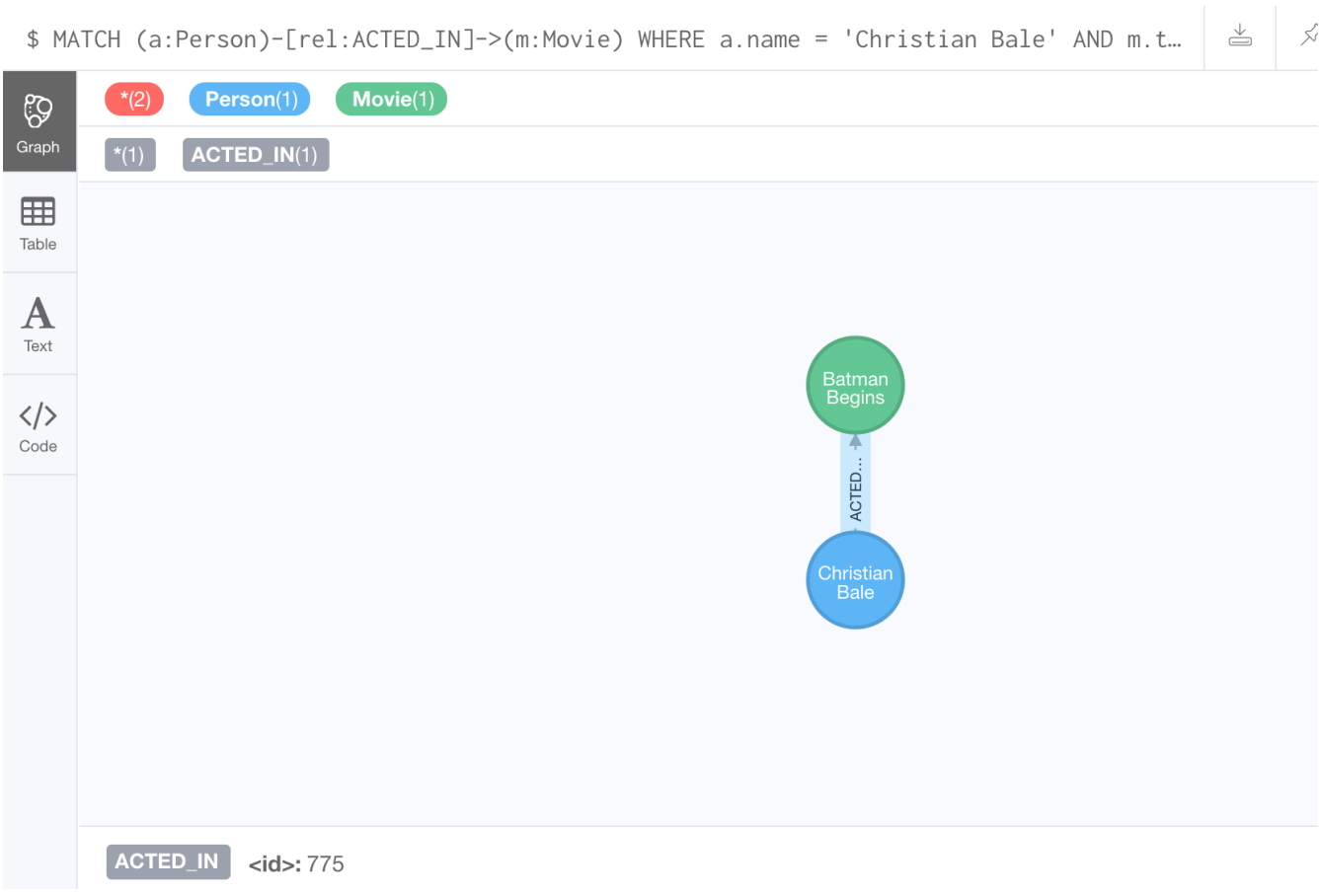
Removing properties from a relationship

There are two ways that you can remove a property from a node. One way is to use the REMOVE keyword. The other way is to set the property's value to `null`, just as you do for properties of nodes.

Suppose we have added the *ACTED_IN* relationship between *Christian Bale* and the movie, *Batman Returns* where the *roles* property is added to the relationship. Here is the code to remove the *roles* property, yet keep the *ACTED_IN* relationship:

```
MATCH (a:Person)-[rel:ACTED_IN]->(m:Movie)
WHERE a.name = 'Christian Bale' AND m.title = 'Batman Begins'
REMOVE rel.roles
RETURN a, rel, m
```

Here is the result returned. An alternative to `REMOVE rel.roles` would be `SET rel.roles = null`



Exercise 9: Creating Relationships

In the query edit pane of Neo4j Browser, execute the browser command: `:play http://guides.neo4j.com/intro-neo4j-exercises` and follow the instructions for Exercise 9.

Deleting nodes and relationships

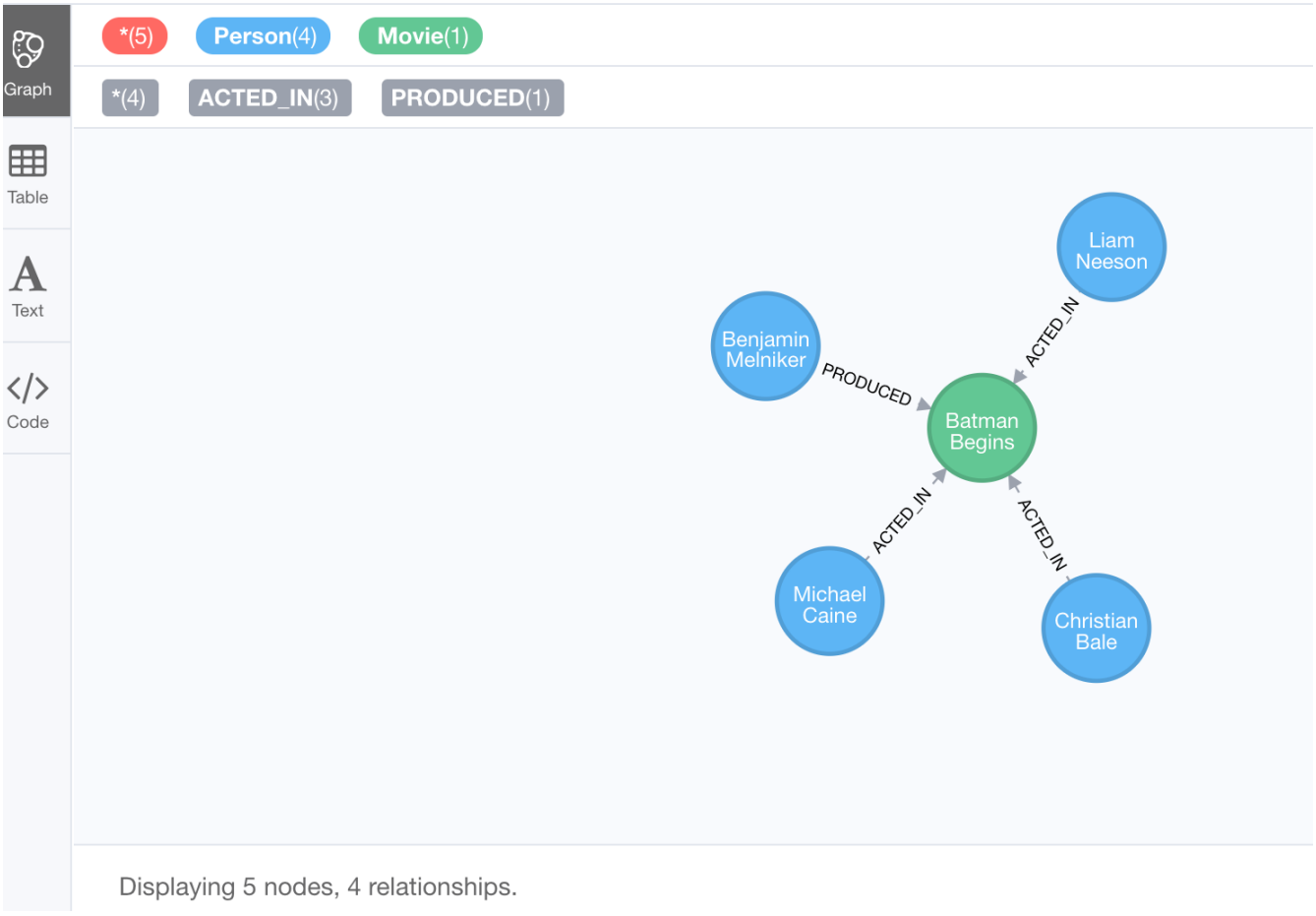
If a node has no relationships to any other nodes, you can simply delete it from the graph using the `DELETE` clause. Relationships are also deleted using the `DELETE` clause.

Note: If you attempt to delete a node in the graph that has relationships in or out of the node, the graph engine will return an error because deleting such a node will leave *orphaned* relationships in the graph.

Deleting relationships

Here are the existing nodes and relationships for the *Batman Begins* movie:

```
$ MATCH (a:Person)-[rel]->(m:Movie) WHERE m.title = 'Batman Begins' RETURN a, rel, m
```



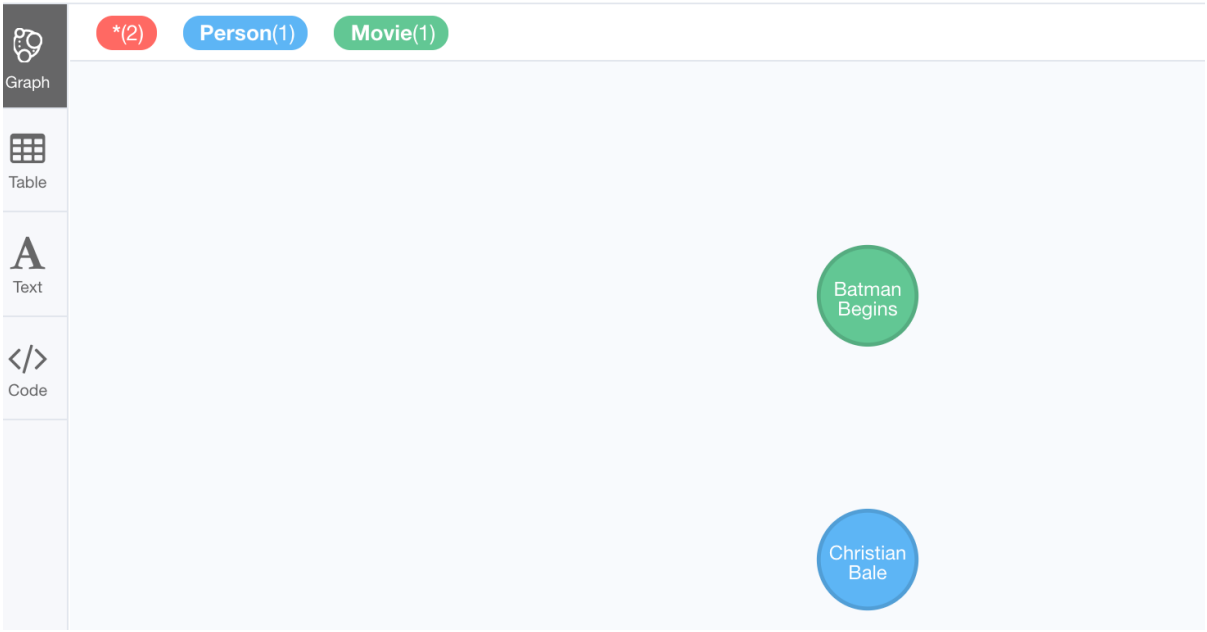
You can delete a relationship between nodes by first finding it in the graph and then deleting it.

In this example, we want to delete the *ACTED_IN* relationship between *Christian Bale* and the movie *Batman Begins*. We find the relationship, and then delete it.

```
MATCH (a:Person)-[rel:ACTED_IN]->(m:Movie)
WHERE a.name = 'Christian Bale' AND m.title = 'Batman Begins'
DELETE rel
RETURN a, m
```

Here is the result of running this Cypher code:

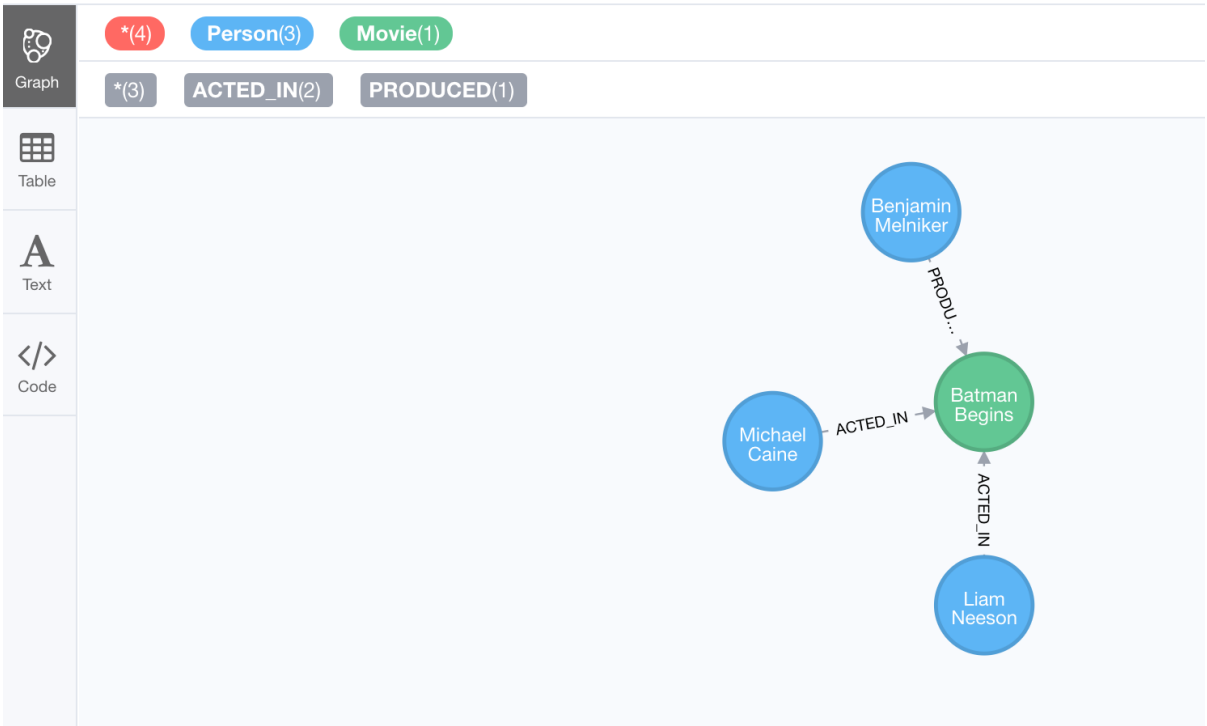
```
$ MATCH (a:Person)-[rel:ACTED_IN]->(m:Movie) WHERE a.name = 'Christian Bale' AND m.t…
```



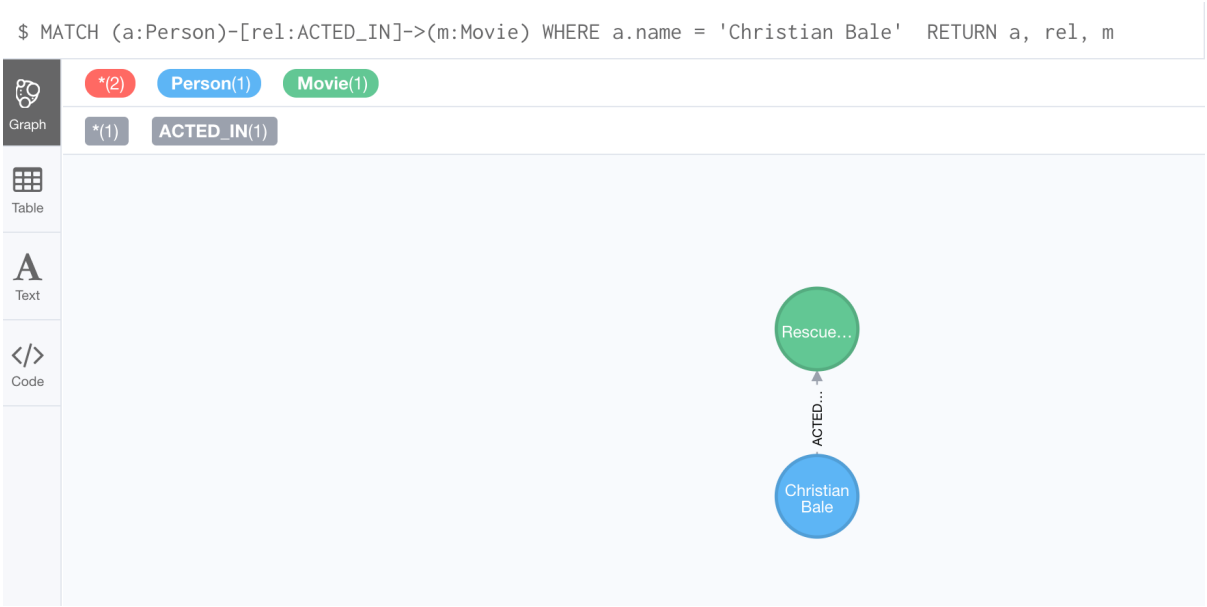
Notice that there no longer exists the relationship between *Christian Bale* and the movie *Batman Begins*.

We can now query the nodes related to *Batman Begins* to see that this movie now only has two actors and one producer connected to it:

```
$ MATCH (a:Person)-[rel]->(m:Movie) WHERE m.title = 'Batman Begins' RETURN a, rel, m
```



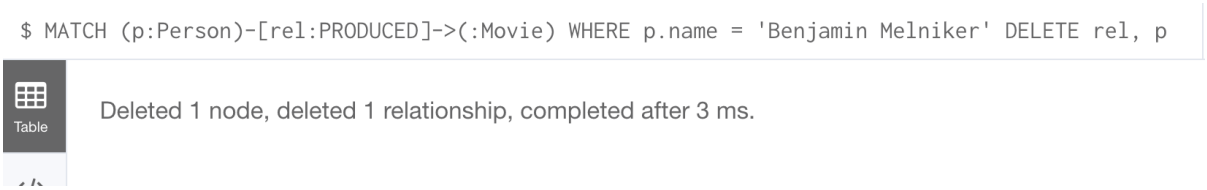
Even though, we have deleted the relationship between actor, *Christian Bale* and the movie *Batman Begins*, we note that this actor is connected to another movie in the graph, so we should not delete this *Christian Bale* node.



In this example, we find the node for the producer, *Benjamin Melniker*, as well as his relationship to movie nodes. First, we delete the relationship(s), then we delete the node:

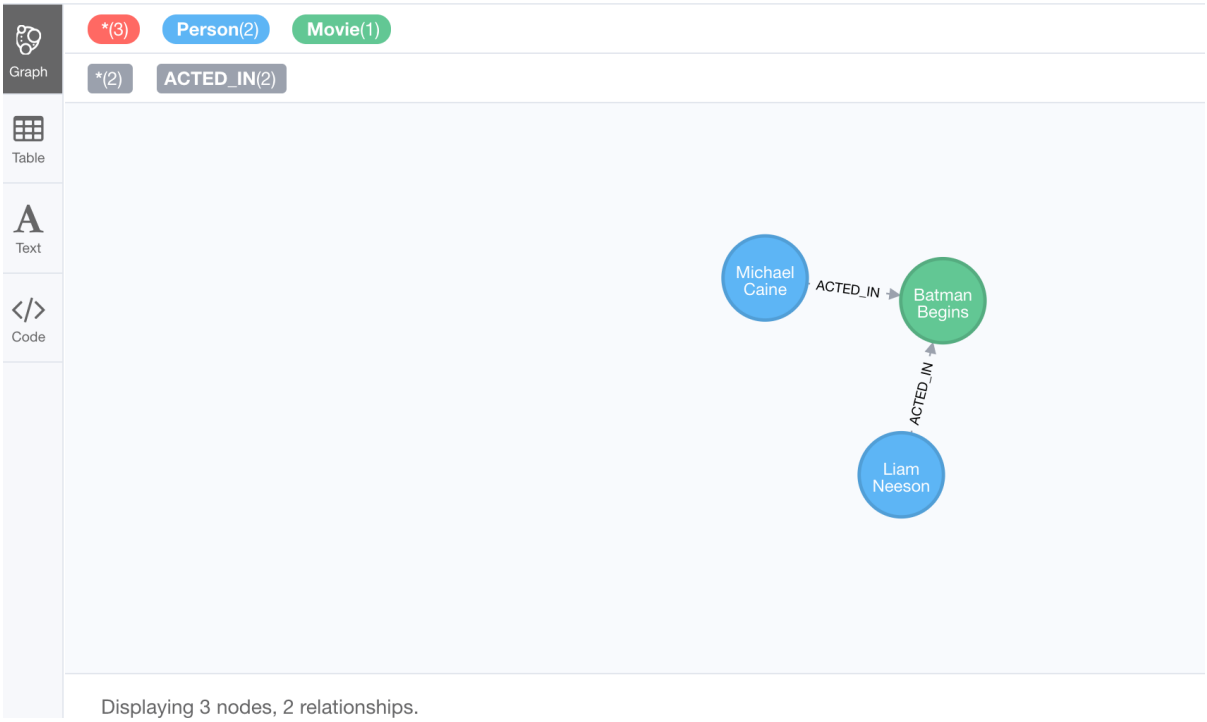
```
MATCH (p:Person)-[rel:PRODUCED]->(:Movie)
WHERE p.name = 'Benjamin Melniker'
DELETE rel, p
```

Here is the result of running this Cypher code:



And here we see that we now have only two connections to the *Batman Begins* movie:

```
$ MATCH (a:Person)-[rel]->(m:Movie) WHERE m.title = 'Batman Begins' RETURN a, rel, m
```



Deleting nodes and relationships

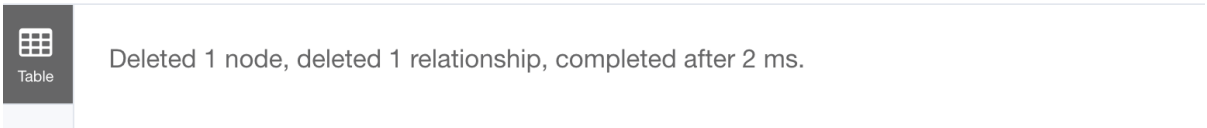
The most efficient way to delete a node and its corresponding relationships is to specify `DETACH DELETE`. When you specify `DETACH DELETE` for a node, the relationships to and from the node are deleted, then the node is deleted.

Here we delete the *Liam Neeson* node and its relationships to any other nodes:

```
MATCH (p:Person)-[:ACTED_IN]->(:Movie)
WHERE p.name = 'Liam Neeson'
DETACH DELETE  p
```

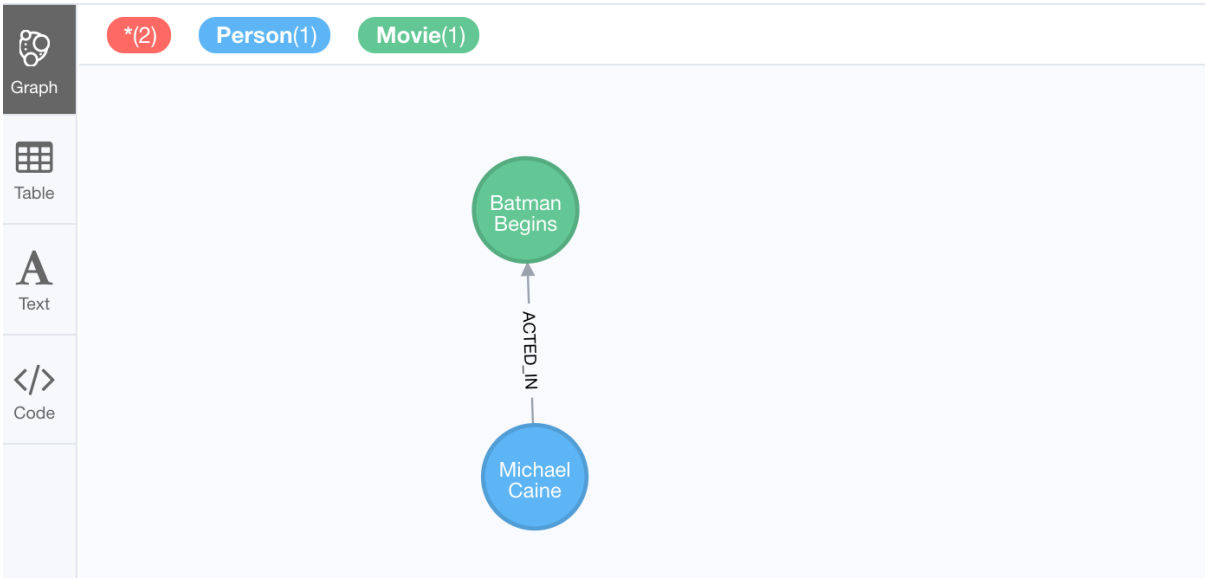
Here is the result of running this Cypher code:

```
$ MATCH (p:Person)-[:ACTED_IN]->(:Movie) WHERE p.name = 'Liam Neeson' DETACH DELETE p
```



And here is what the *Batman Begins* node and its relationships now look like. There is only one actor, *Michael Caine* connected to the movie.

```
$ MATCH (a:Person)--(m:Movie) WHERE m.title = 'Batman Begins' RETURN a, m
```



Exercise 10: Deleting Nodes and Relationships

In the query edit pane of Neo4j Browser, execute the browser command: `:play http://guides.neo4j.com/intro-neo4j-exercises` and follow the instructions for Exercise 10.

Merging data in the graph

Thus far, you have learned how to create nodes, labels, properties, and relationships in the graph. How the graph engine behaves when a duplicate element is created depends on the type of element:

If you use `CREATE`...	The result is...
Node	If a node with the same property values exists, a duplicate node is created.
Label	If the label already exists for the node, the node is not updated.
Property	If the node or relationship property already exists, it is updated with the new value. **Note:** If you specify a set of properties to be created, it could remove existing properties if they are not included in the set.
Relationship	If the relationship exists, a duplicate relationship is created.

Important: You should never create duplicate nodes or relationships in a graph.

The `MERGE` clause is used to find elements in the graph. If the element is not found, it is created.

You use the `MERGE` clause to:

- Create a node with a different label (You do not want to add a label to an existing node.).
- Create a node with a different set of properties (You do not want to update a node with existing properties.).
- Create a unique relationship between two nodes.

Using `MERGE` to create nodes

Here is the simplified syntax for the `MERGE` clause for a node:

```
MERGE (`variable`:`label`{node-properties}) |
MERGE (`variable`:`label`{node-properties})-[`relationship`]-(`other-node`)
RETURN `variable`
```

If there is an existing node with `*label*` and `*node-properties*` or with the `*relationship*` to `*other-node*` found in the graph, no node is created. If, however the node is not found in the graph, then the node is created.

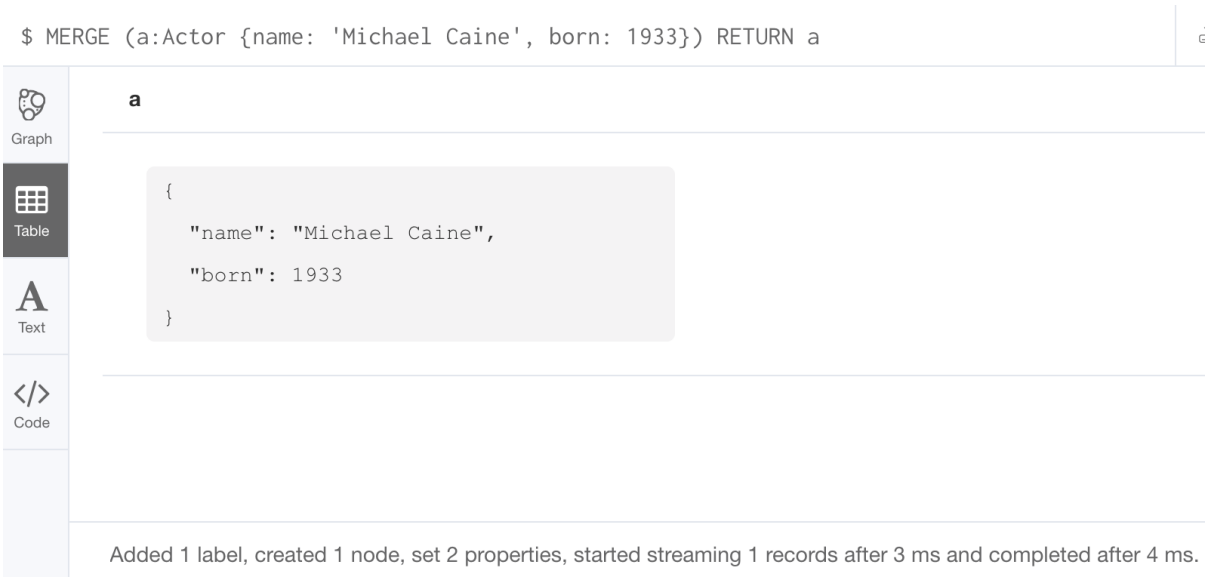
Here is what we currently have in the graph for the *Person*, *Michael Caine*. This node has values for *name* and *born*. Notice also that the label for the node is *Person*.

```
$ MATCH (a:Person {name: 'Michael Caine', born: 1933}) RETURN a
```

Here we use `MERGE` to find a node with the *Actor* label with the properties *name* of *Michael Caine*, and *born* of *1933*.

```
MERGE (a:Actor {name: 'Michael Caine', born: 1933})
RETURN a
```

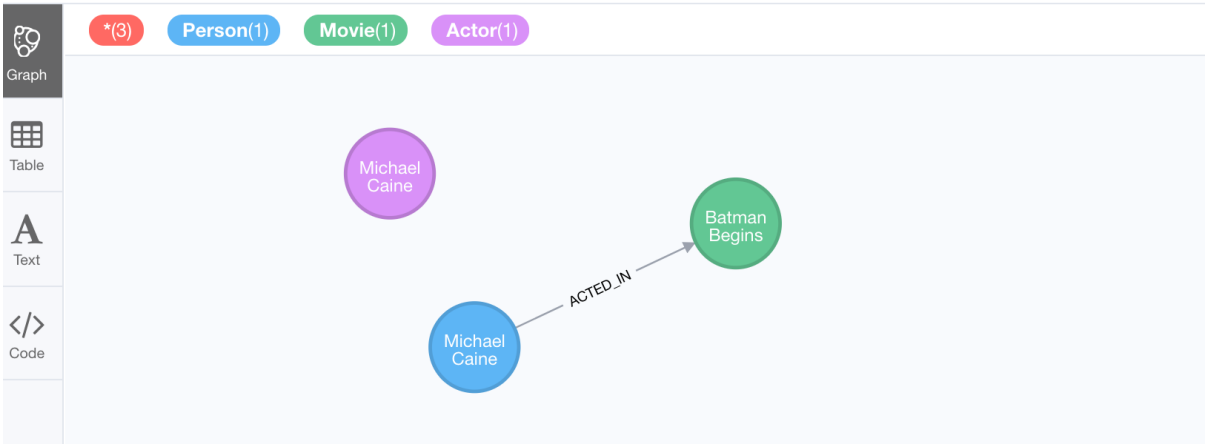
Here is the result of running this Cypher code. We do find a node with the label *Actor* so the graph engine creates one.



If we were to repeat this `MERGE` clause, no additional nodes would be created in the graph.

At this point, however, we have two *Michael Caine* nodes in the graph, one of type *Person*, and one of type *Actor*:

```
$ MATCH (a {name: 'Michael Caine'}), (m:Movie) WHERE m.title = 'Batman Begins' RETURN a, m
```



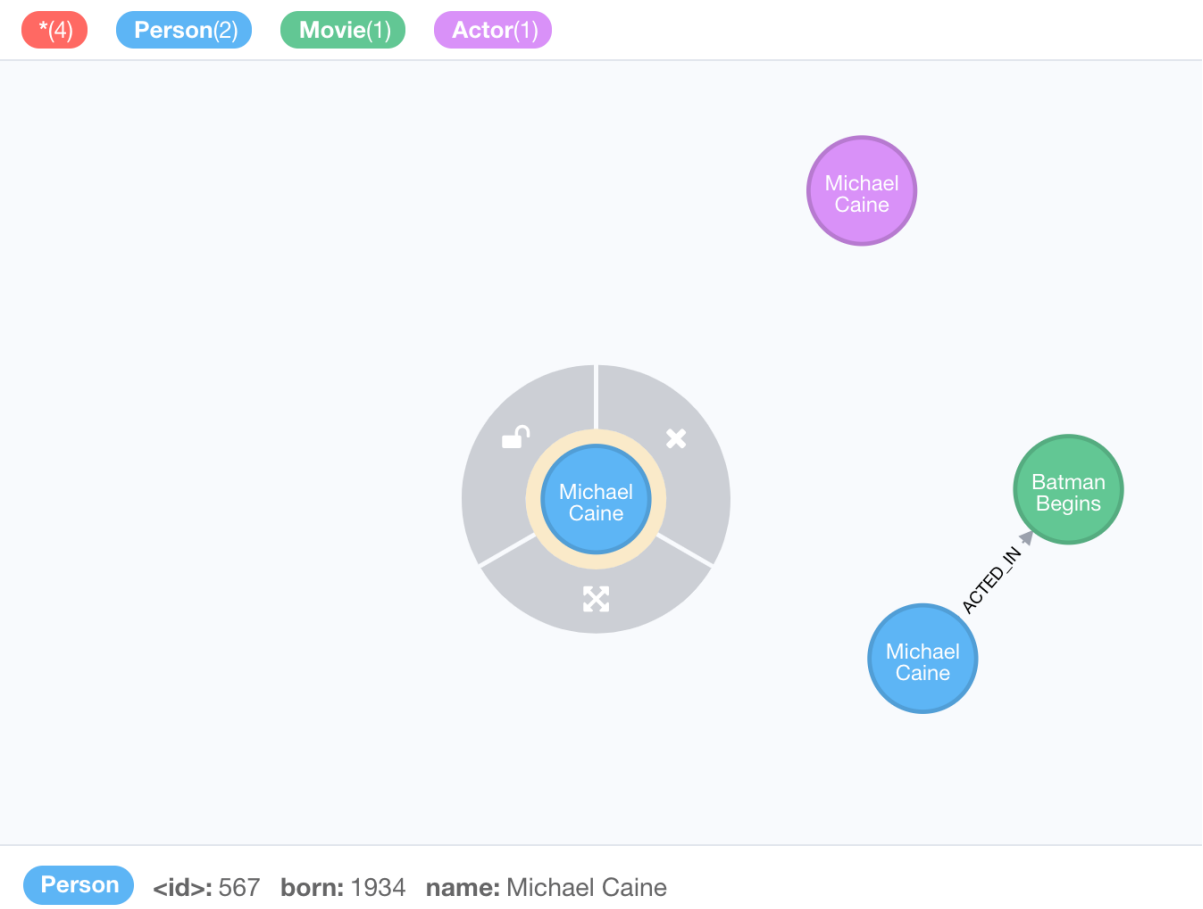
A similar behavior occurs when you specify the same label, but the set of properties for the node differ. If the node does not exist with the exact same set of properties with their values, the graph engine creates a new node.

Here is an example where we create a *Person* node for a person with the values *Michael Caine* and *1934* respectively for *name* and *born*.

```
MERGE (a:Person {name: 'Michael Caine', born: 1934})
RETURN a
```

We already have a *Person* node for *Michael Caine* with a *born* value, *1933*. Since the set of properties do not match what is in the graph, the graph engine creates a new node.

Here are *Michael Caine* nodes in our graph after this `MERGE`:



Important: Be mindful that node labels and the properties for a node are significant when merging nodes.

Specifying creation behavior when merging

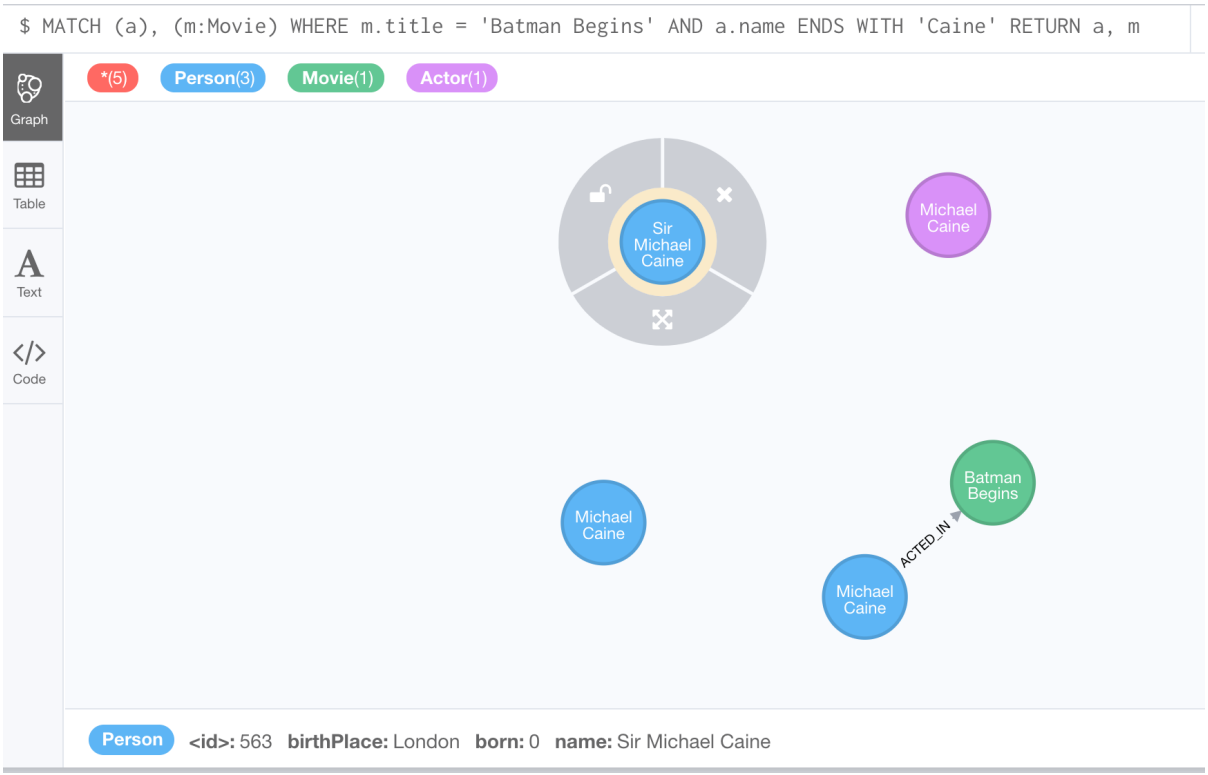
An alternative to specifying the values for properties when you specify the `MERGE` clause, you can use the `MERGE` clause, along with `ON CREATE` to assign specific values to a node being created as a result of a merge attempt.

Here is an example where create a new node, specifying property values for the new node.

```
MERGE (a:Person {name: 'Michael Caine', born: 0})
ON CREATE SET a.name = 'Sir Michael Caine',
              a.birthPlace = 'London'
RETURN a
```

We know that the existing *Michael Caine Person* nodes exist and *born* values of *1933* and *1934*. When the `MERGE` executes, it will not find any matching nodes and will execute the code in the `ON CREATE` clause where we set the *name* and *birthPlace* property values. The value for *born* will be set to *0*.

Here is the resulting nodes that have anything to do with *Michael Caine*. The most recently created node has the *name* value of *Sir Michael Caine*.



You can also specify an `ON MATCH` clause during merge processing. If the node is found, you can update its properties. Here is an example:

```
MERGE (a:Person {name: 'Michael Caine', born: 0})
ON CREATE SET a.name = 'Sir Michael Caine',
              a.birthPlace = 'London'
ON MATCH SET a.birthPlace = 'London'
RETURN a
```

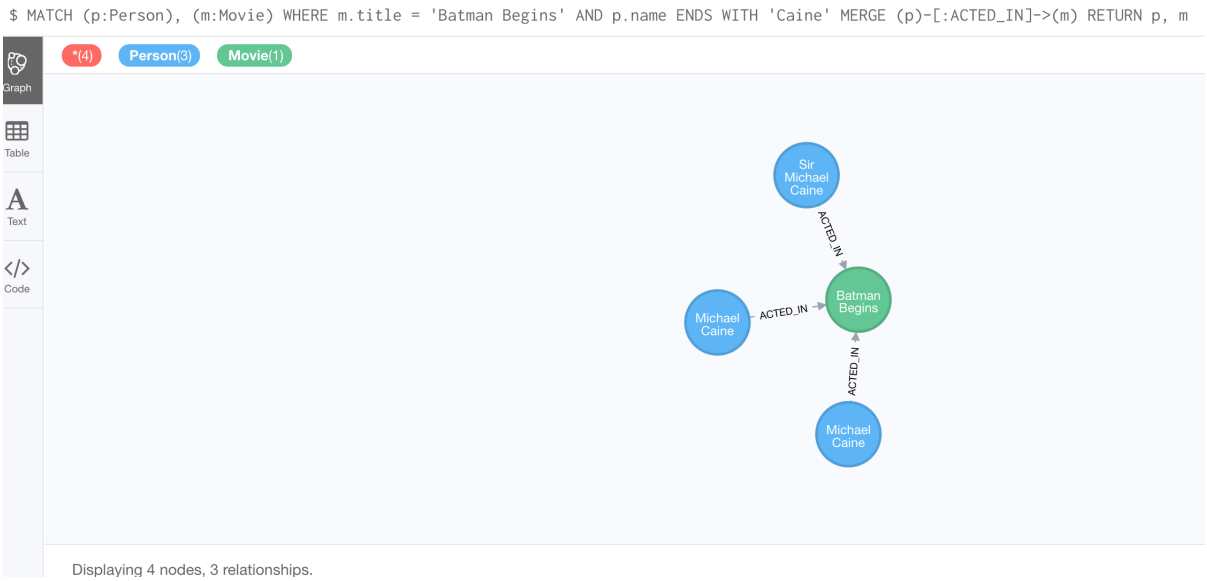
Using `MERGE` to create relationships

Using `MERGE` to create relationships is something you should always do because it ensures that all relationships in the graph will be unique.

In this example, we use the `MATCH` clause to find all *Person* nodes that represent *Micheal Caine* and we find the movie, *Batman Begins* that we want to connect to all of these nodes. We already have a connection between one of the *Person* nodes and the *Movie* node. We do not want this relationship to be duplicated. This is where we can use `MERGE` as follows:

```
MATCH (p:Person), (m:Movie)
WHERE m.title = 'Batman Begins' AND p.name ENDS WITH 'Caine'
MERGE (p)-[:ACTED_IN]->(m)
RETURN p, m
```

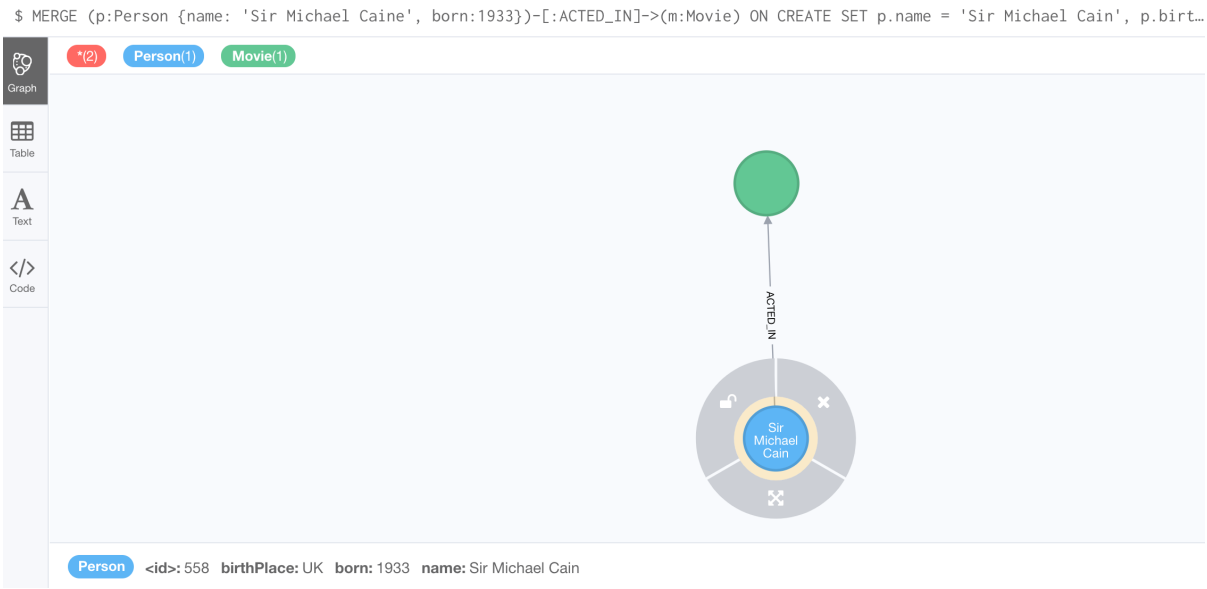
Here is the result of executing this Cypher code. It went through all the nodes and added the relationship to the nodes that didn't already have the relationship.



You must be aware of the behavior of the `MERGE` clause and how it will automatically create nodes and relationships. Suppose you wanted to create a new *Person* node for *Sir Michael Caine* who acted in a movie. If we specify this `MERGE` clause in our code, we will find that, not only does it create a new *Person* node, but also creates the relationship to another new *Movie* node. Here is our code:

```
MERGE (p:Person {name: 'Sir Michael Caine', born:1933})-[:ACTED_IN]->(m:Movie)
ON CREATE SET p.name = 'Sir Michael Cain', p.birthPlace = 'UK'
RETURN p, m
```

Here is the result of executing this Cypher code. Notice that it created two nodes and the relationship.



Important: Use caution when using `MERGE` to automatically create nodes and relationships.

Exercise 11: Merging Data in the Graph

In the query edit pane of Neo4j Browser, execute the browser command: `:play http://guides.neo4j.com/intro-neo4j-exercises` and follow the instructions for Exercise 11.

Check your understanding

Question 1

What Cypher clauses can you use to create a node?

Select the correct answers.

- ▢ CREATE
- ▢ CREATE NODE
- ▢ MERGE
- ▢ ADD

Question 2

Suppose that you have retrieved a node,*n* with a property, *color*:

```
MATCH (s:Shape {location: [20,30]})
???
RETURN s
```

What Cypher code do you add here to delete the *color* property from this node?

Select the correct answers.

- ▢ DELETE s.color
- ▢ SET s.color=null
- ▢ REMOVE s.color
- ▢ SET s.color=?

Question 3

Suppose you retrieve a node, *n* in the graph that is related to other nodes. What Cypher code do you write to delete this node and its relationships in the graph?

Select the correct answers.

- ▢ DELETE n
- ▢ DELETE n WITH RELATIONSHIPS
- ▢ REMOVE n
- ▢ DETACH DELETE n

Summary

You should now be able to write Cypher code to:

- Create a node
  - Add and remove node labels
  - Add and remove node properties
  - Update properties
- Create a relationship
  - Add and remove properties for a relationship
- Delete a node
- Delete a relationship
- Merge data in a graph
  - Creating nodes
  - Creating relationships