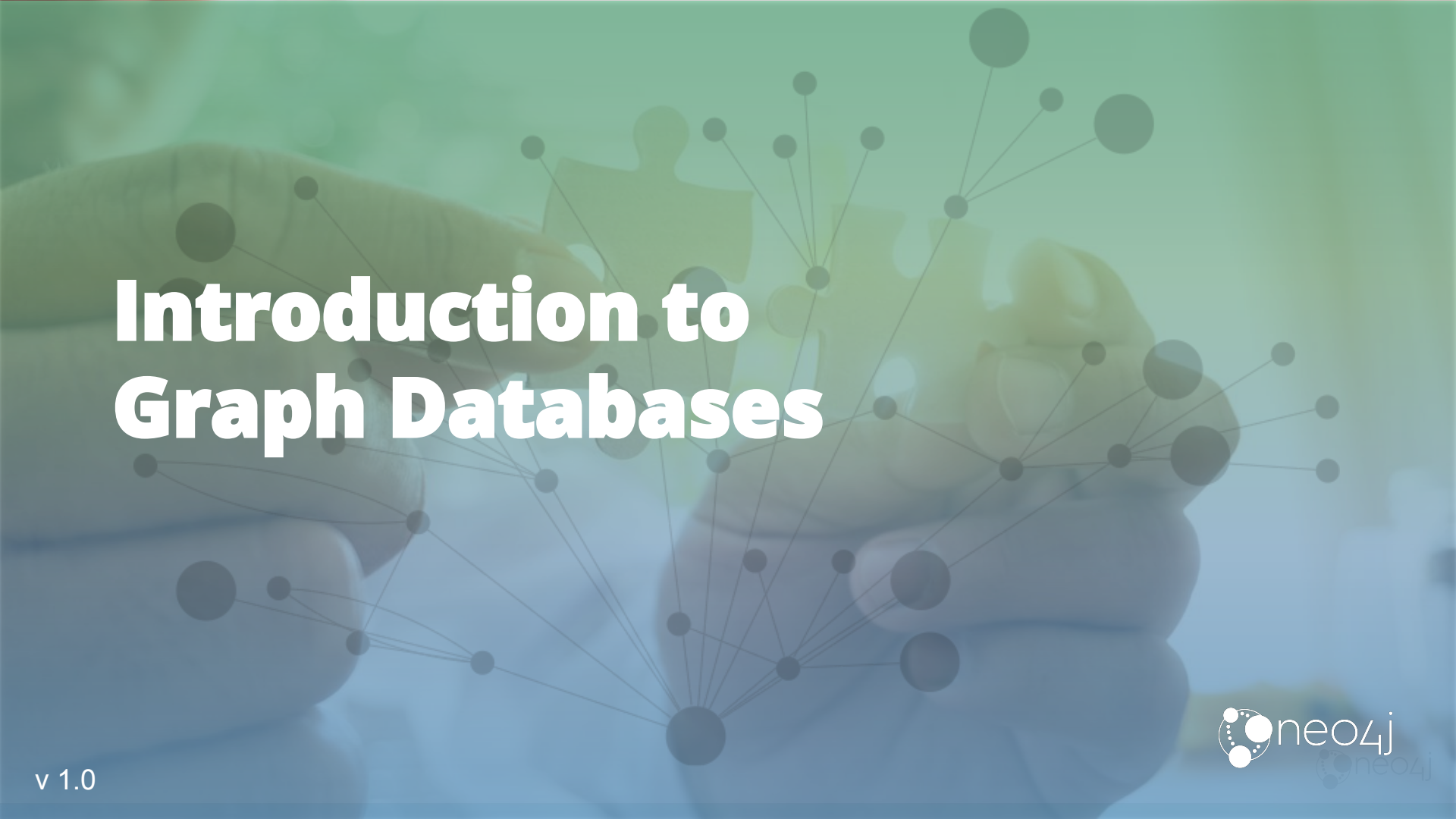

Introduction to
Graph Databases
v 1.0
neo4j

Overview

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

- Describe what a graph database is.
- Describe some common use cases for using a graph database.
- Describe how real-world scenarios are modeled as a graph.

Evolution of graph databases

Tiny RAM

Spinning Platters
(Low Capacity /
Slow, Sequential IO)

The RDBMS Era

Relational Model

RDBMS

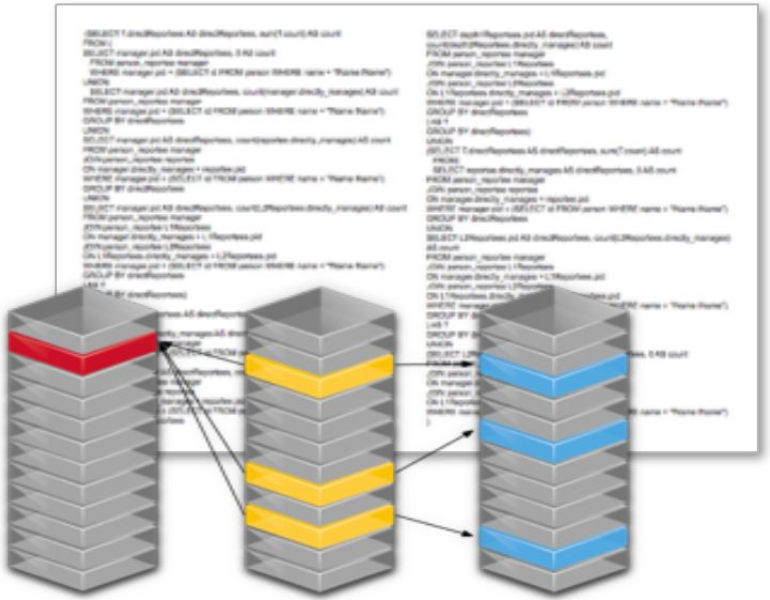
Relational databases cannot handle relationships

1 **Wrong Model**
They cannot model or store relationships without complexity

2 **Degraded Performance**
Speed plummets as data grows and as the number of joins grows

3 **Wrong Language**
SQL was built with Set Theory in mind, not Graph Theory

4 **Not Flexible**
New types of data and relationships require schema redesign



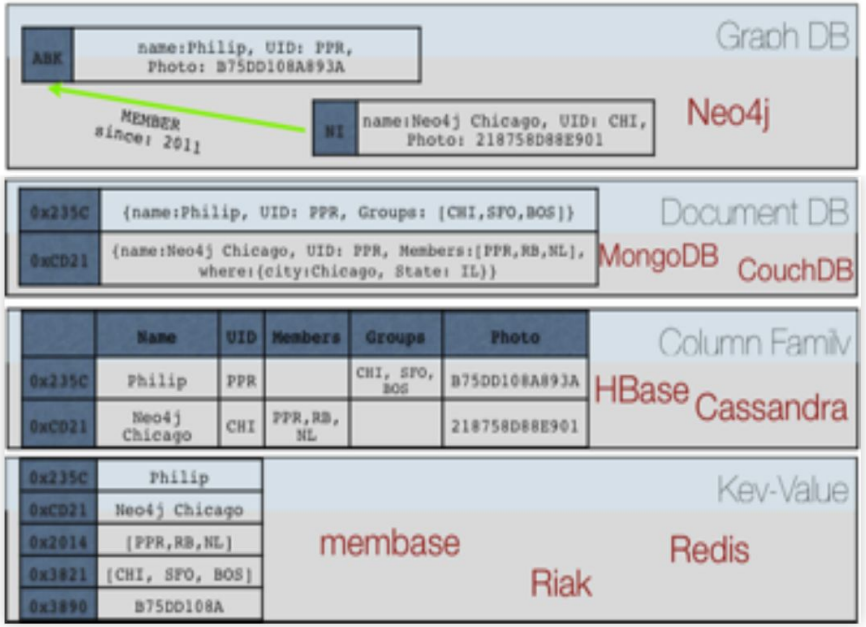
NoSQL databases cannot handle relationships

1 Wrong Model
They cannot model or store relationships without complexity

2 Degraded Performance
Speed plummets as you try to join data together in the application

3 Wrong Languages
Lots of wacky “almost sql” languages terrible at “joins”

4 Not ACID
Eventually Consistent means Eventually Corrupt



neo4j

What is a graph database?

1 Right Model
Graphs simplify how you think

2 Better Performance
Query relationships in real time

3 Right Language
Cypher was purpose built for Graphs

4 Flexible and Consistent
Evolve your schema seamlessly while keeping transactions

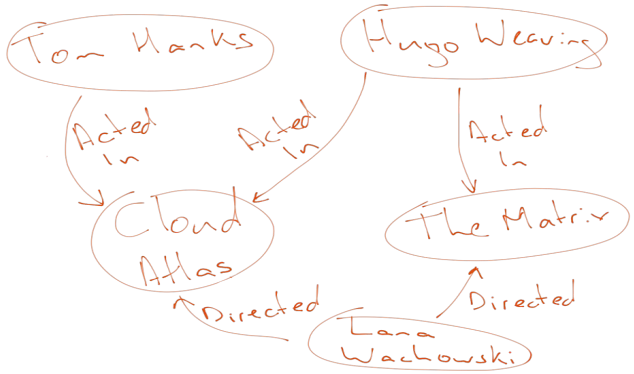
Agile, High Performance and Scalable without Sacrifice

ACID

neo4j

The case for graph databases

- Intuitiveness
  - Create and maintain data in a logical fashion
  - Lessening the translation “friction”
  - Whiteboard model is the physical model
- Speed
  - Development
  - Execution
- Agility
  - Naturally adaptive, schema optional database
- Cypher query language for graphs
  - Less time writing queries
  - More time asking the next questions about the data



SQL vs Cypher

SQL Query

Cypher Query

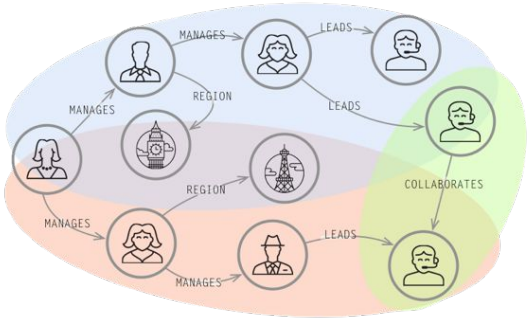
```
MATCH (boss)-[:MANAGES*0..3]->(sub),
      (sub)-[:MANAGES*1..3]->(report)
WHERE boss.name = “John Doe”
RETURN sub.name AS Subordinate,
  count(report) AS Total
```

Find all direct reports and
how many people they manage,
up to 3 levels down

Use cases

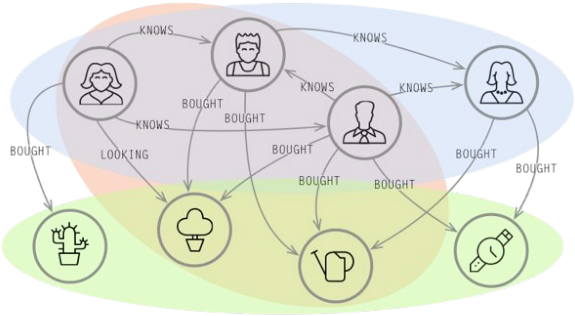
Internal Applications

- Master data management
- Network and IT operations
- Fraud detection



Customer-Facing Applications

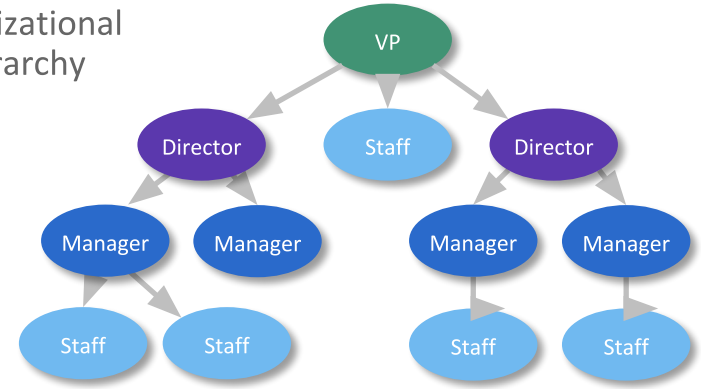
- Real-Time Recommendations
- Graph-Based Search
- Identity and Access Management



Use case: Real-time recommendations

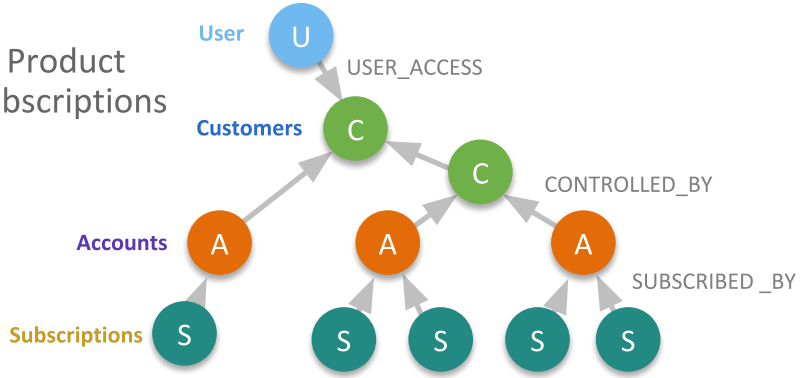
Use cases: Master data management.

Organizational Hierarchy

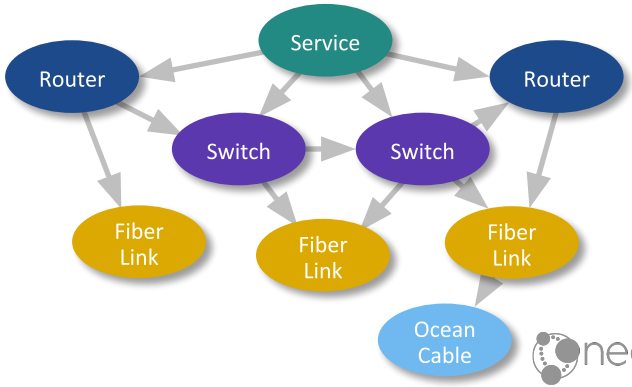


Customer 360

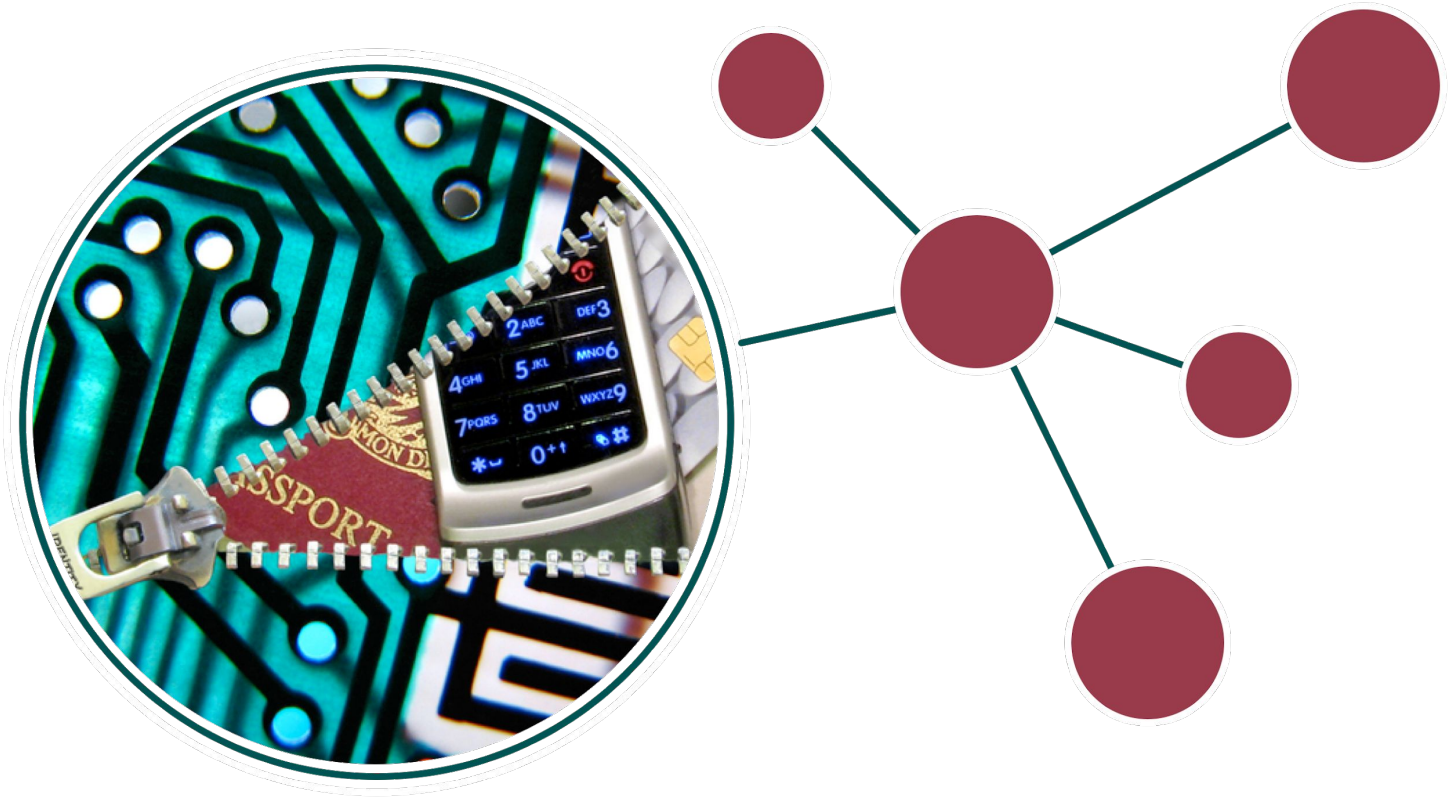
Product Subscriptions



CMDB Network Inventory



Use case: Fraud detection

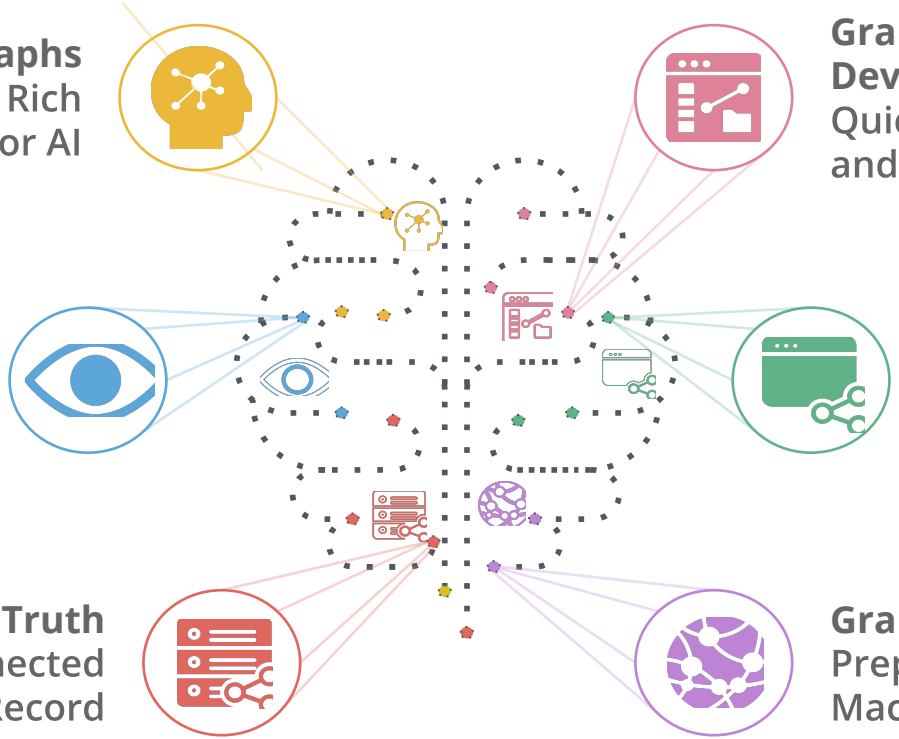


Use case: Graph-based search

Knowledge Graphs
Provide Rich Context for AI

Visibility
Human-Friendly Graph Visualization

Connected Truth
Maintain Connected Systems of Record

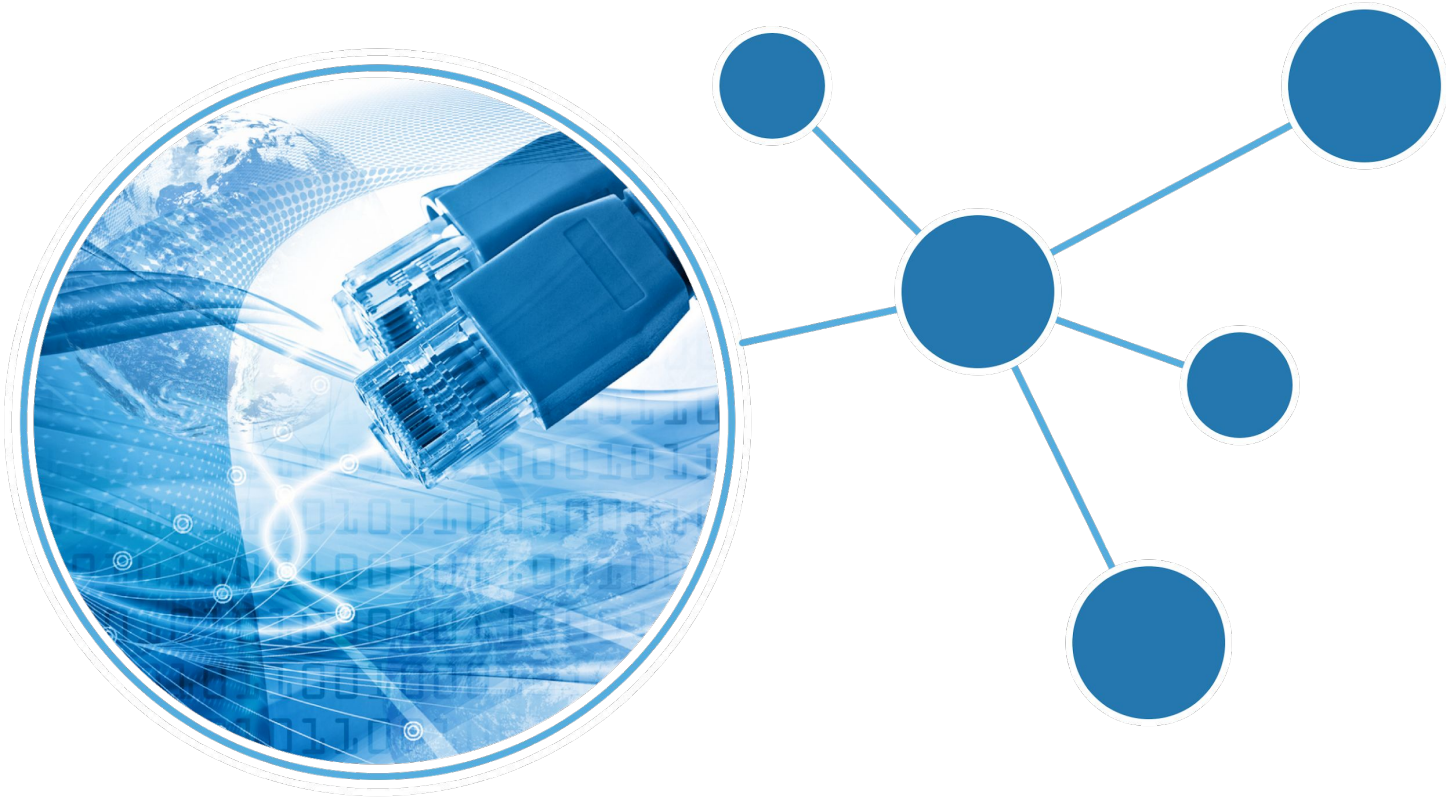


Graph Accelerated Development
Quickly Evaluate Datasets and Features for Extraction

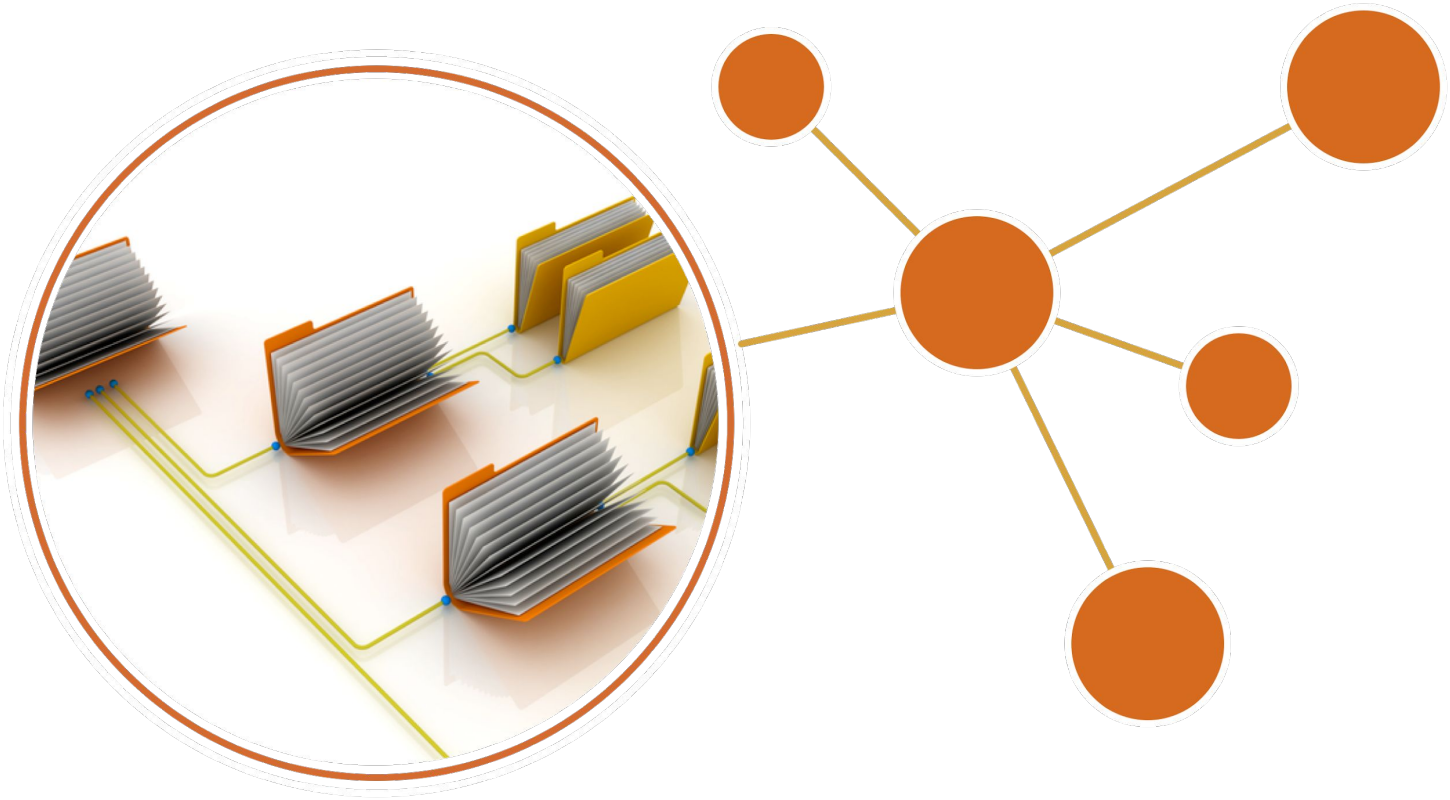
Graph Execution
Operationalize Real-Time OLAP and Monitoring

Graph Enriched Data
Preprocess and Augment Machine Learning Data

Use case: Network and IT operations



Use case: Identity and access management



What is a graph?

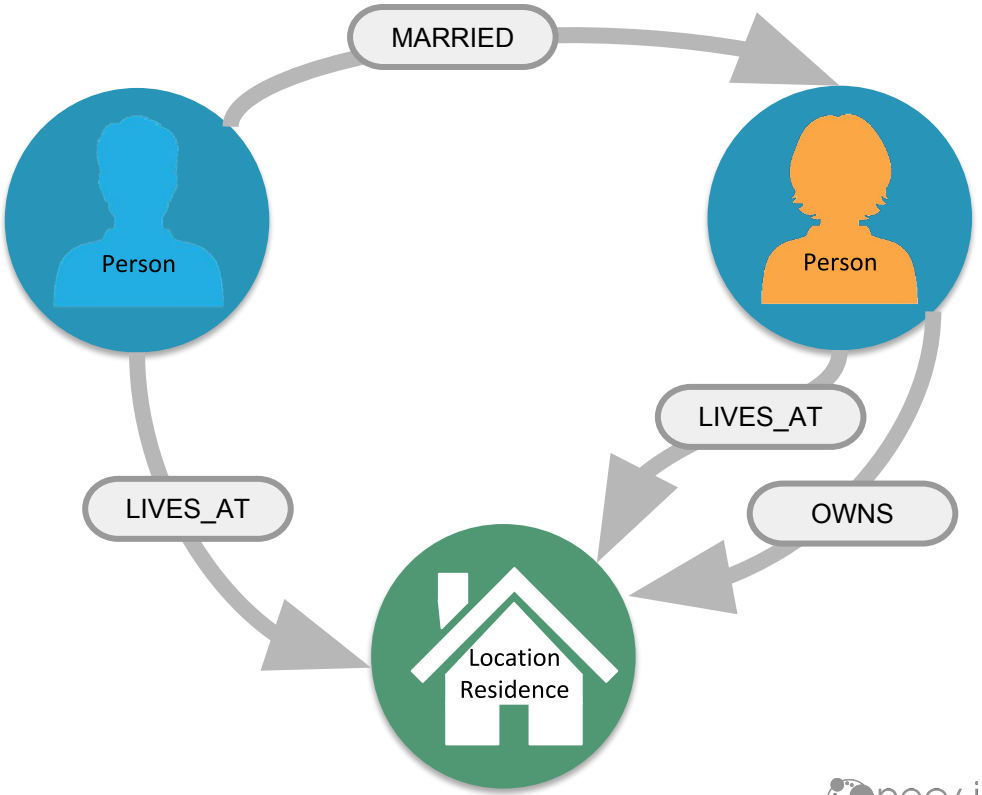
- Nodes
- Relationships
- Properties
- Labels

Nodes

- Nouns in your model
- Represent the objects or entities in the graph
- Can be *labeled*:
  - Person
  - Location
  - Residence
  - Business

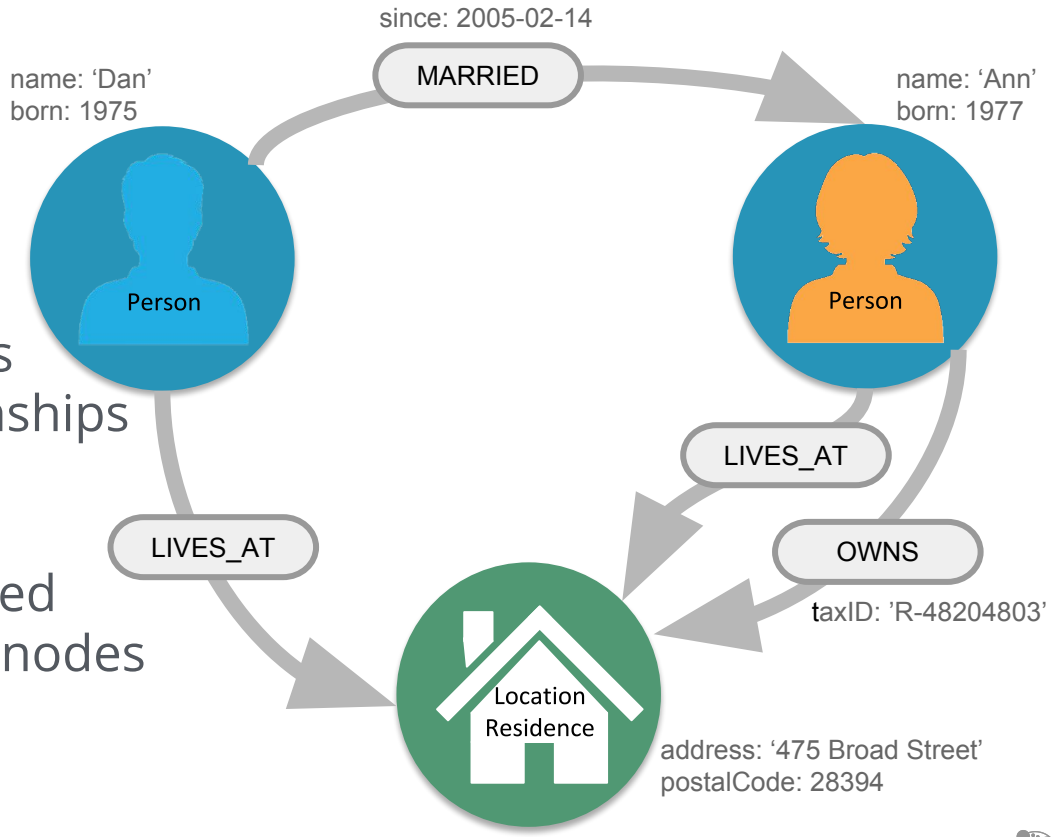
Relationships

- Verbs in your model
- Represent the connection between nodes in the graph
- Has a type:
  - MARRIED
  - LIVES_AT
  - OWNS
- Directed relationship

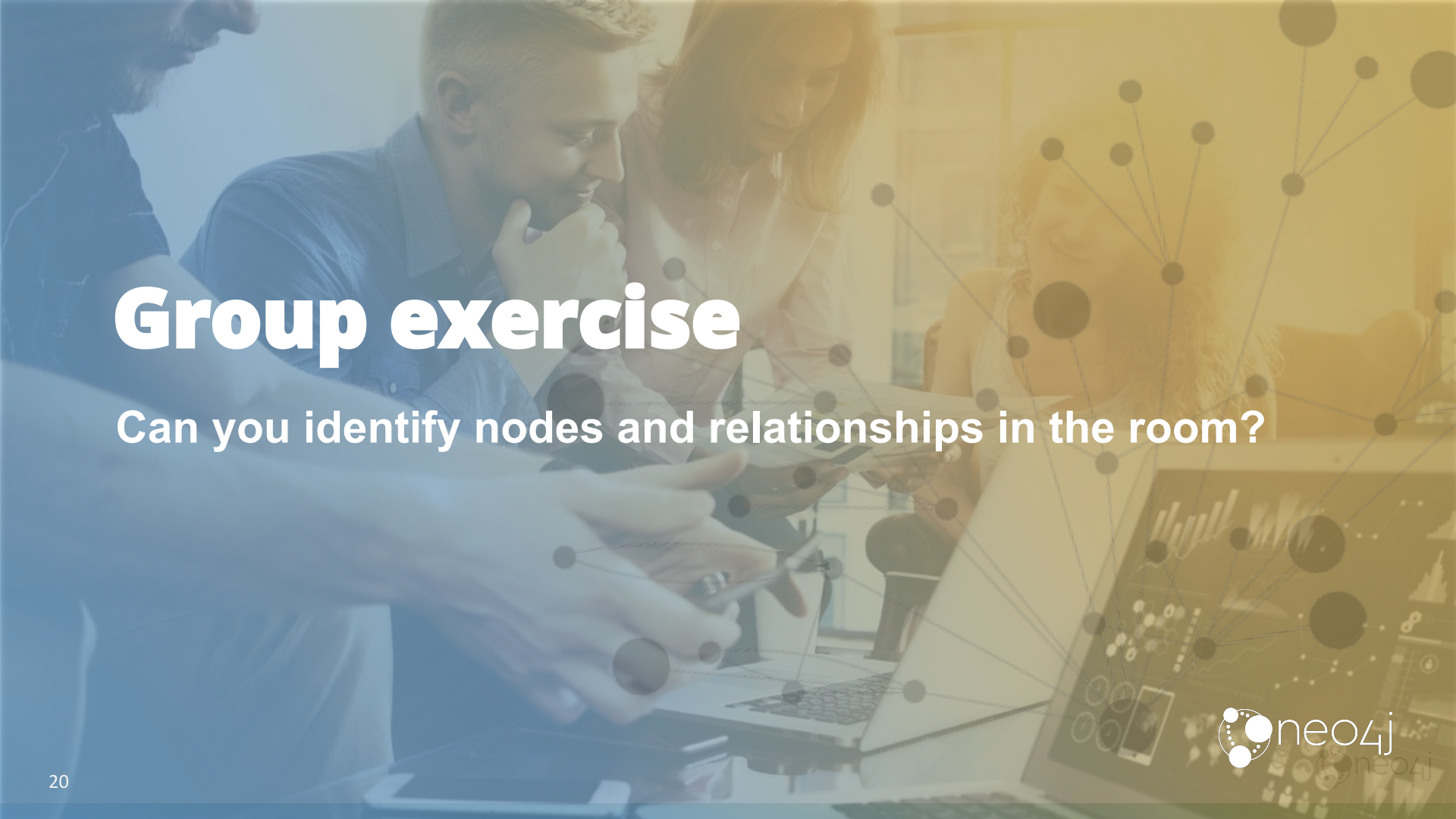


Properties

- Adjectives to describe nodes
- Adverbs to describe relationships
- Property:
  - Key/value pair
  - Can be optional or required
  - Values can be unique for nodes
  - Values have no type



neo4j



Group exercise

Can you identify nodes and relationships in the room?

Modeling relational to graph

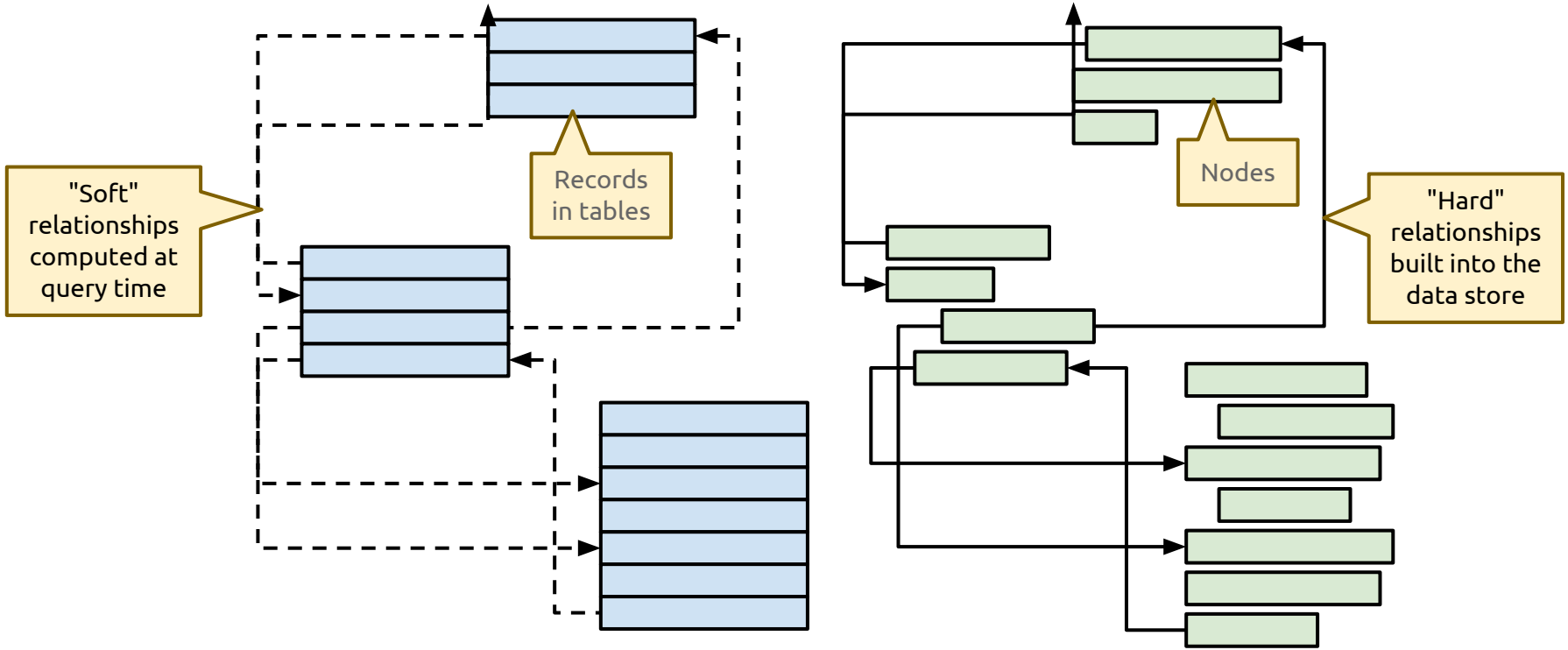
In some ways they're similar:

Relational	Graph
Rows	Nodes
Joins	Relationships
Table names	Labels
Columns	Properties

In some ways they're not:

Relational	Graph
Each column must have a field value.	Nodes with the same label aren't required to have the same set of properties.
Joins are calculated at query time.	Relationships are stored on disk when they are created.
A row can belong to one table.	A node can have many labels.

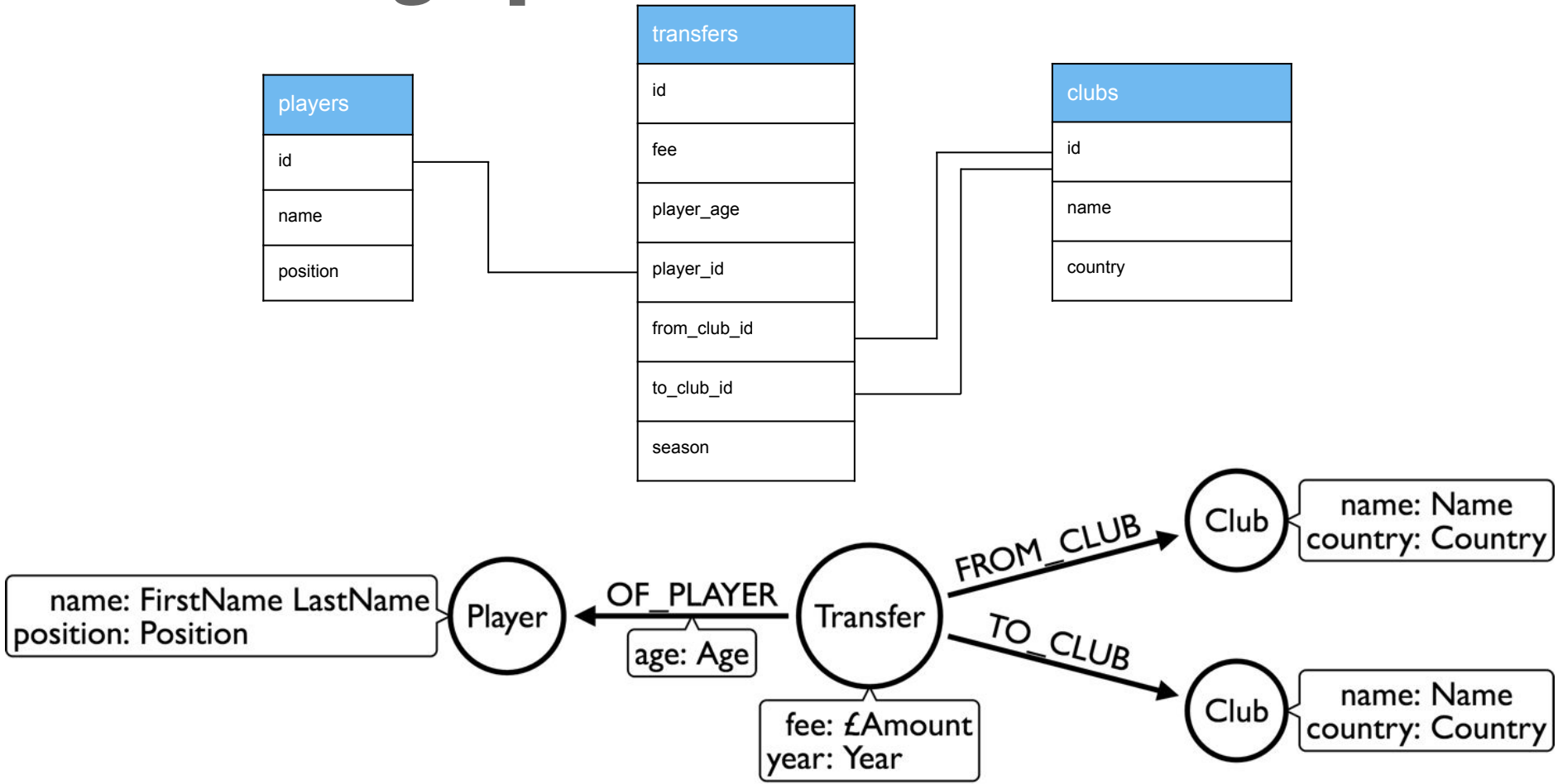
Run-time behavior: RDBMS vs graph



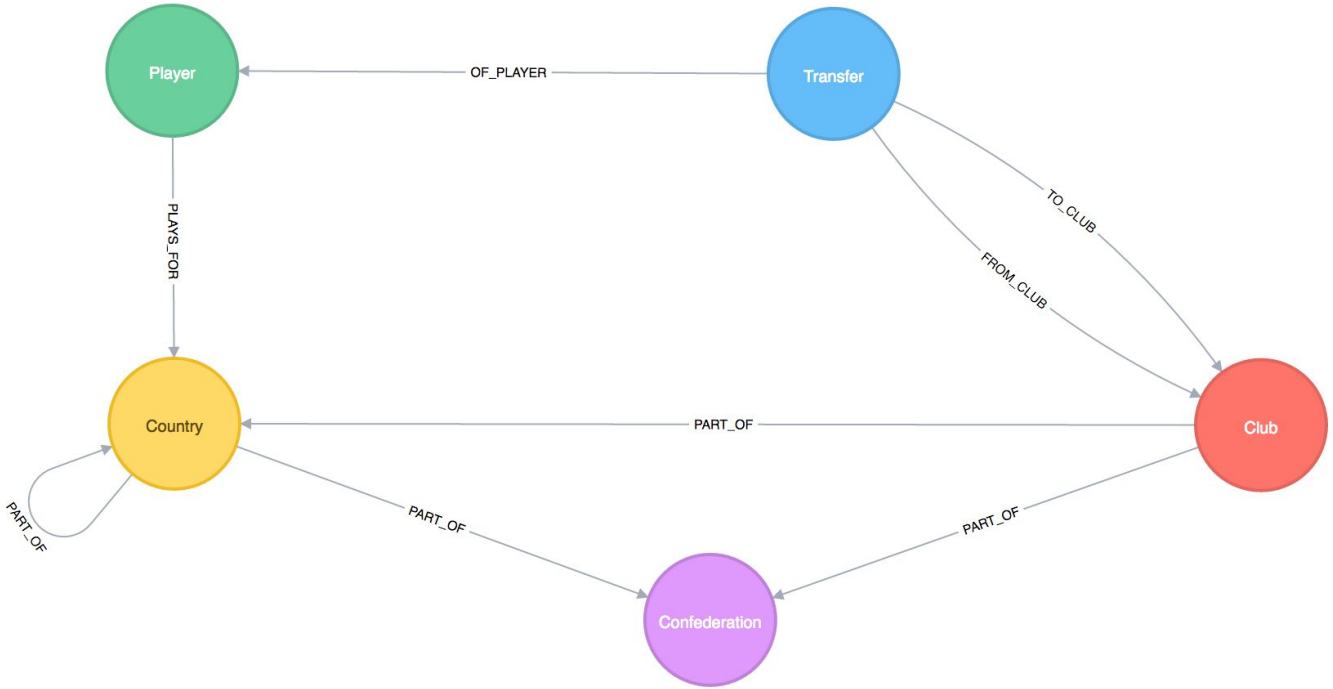
How we model: RDBMS vs graph

Relational	Graph
Try and get the schema defined and then make minimal changes to it after that.	It's common for the schema to evolve with the application.
More abstract focus when modeling. i.e. Focus on classes rather than objects.	Common to use actual data items when modeling.

RDBMS vs graph models



Neo4j data model: CALL db.schema()



How does Neo4j support the property graph model?

- Neo4j is a **Database** - use it to reliably **store information** and **find it later**.
- Neo4j's data model is a **Graph**, in particular a **Property Graph**.
- **Cypher** is Neo4j's graph query language (**SQL for graphs!**).
- Cypher is a declarative query language: it describes **what** you are interested in, not **how** it is acquired.
- Cypher is meant to be very **readable** and **expressive**.


Check your understanding

Question 1

What elements make up a graph?

Select the correct answers.

- ❑ tuples
- ❑ nodes
- ❑ documents
- ❑ relationships

Answer 1

What elements make up a graph?

Select the correct answers.

- ❏ tuples
- ✓ nodes
- ❏ documents
- ✓ relationships

Question 2

Suppose that you want to create a graph to model customers, products, what products a customer buys, and what products a customer rated. You have created nodes in the graph to represent the customers and products. In this graph, what relationships would you define?

Select the correct answers.

- ❑ BOUGHT
- ❑ IS_A_CUSTOMER
- ❑ IS_A_PRODUCT
- ❑ RATED

Answer 2

Suppose that you want to create a graph to model customers, products, what products a customer buys, and what products a customer rated. You have created nodes in the graph to represent the customers and products. In this graph, what relationships would you define?

Select the correct answers.

- ✓ BOUGHT
- ❏ IS_A_CUSTOMER
- ❏ IS_A_PRODUCT
- ✓ RATED

Question 3

What query language is used with a Neo4j Database?

Select the correct answer.

- ❑ SQL
- ❑ CQL
- ❑ Cypher
- ❑ OPath

Answer 3

What query language is used with a Neo4j Database?

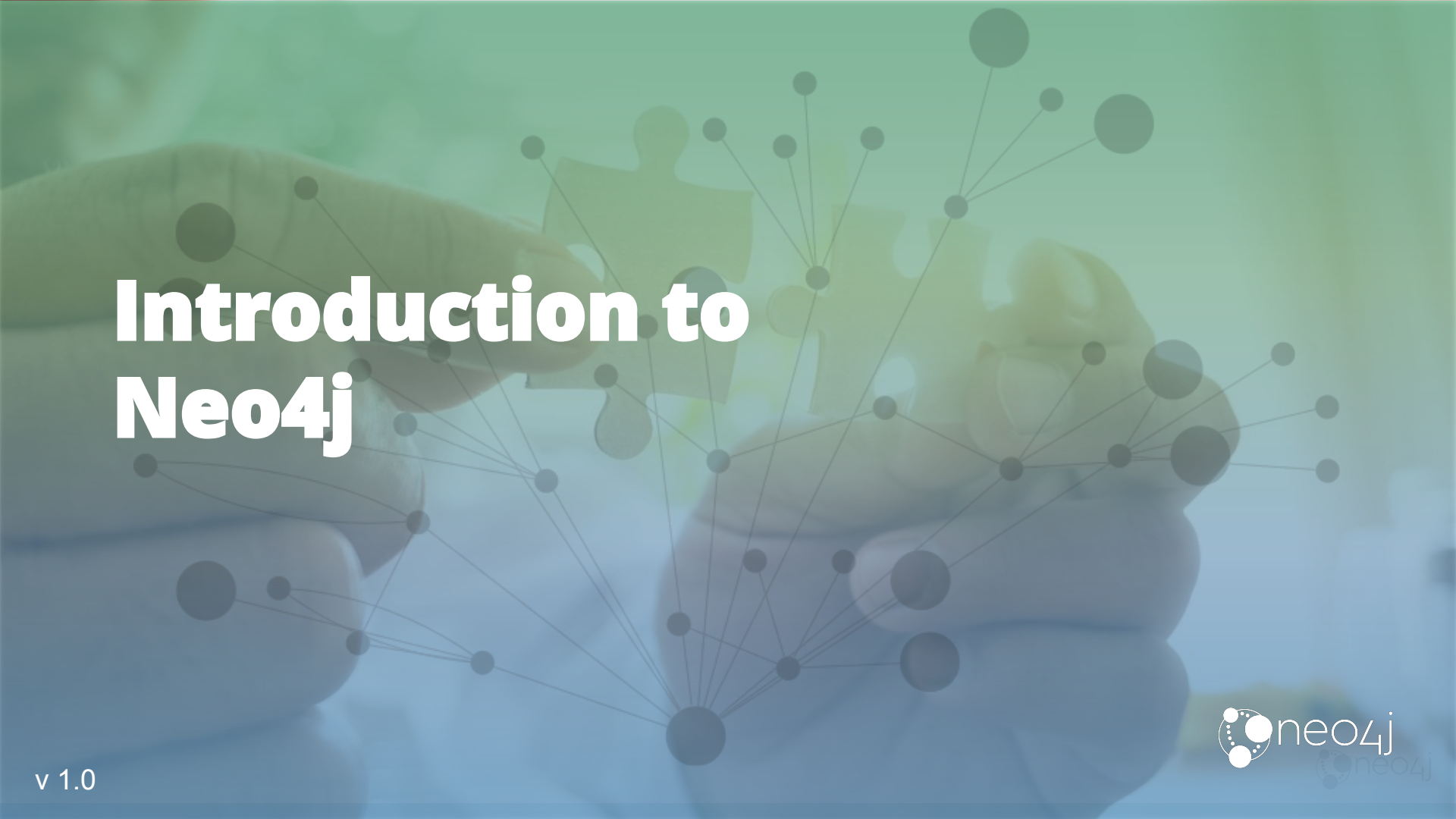
Select the correct answer.

- ❑ SQL
- ❑ CQL
- ✓ Cypher
- ❑ OPath

Summary

You should be able to:

- Describe what a graph database is.
- Describe some common use cases for using a graph database.
- Describe how real-world scenarios are modeled as a graph.


Introduction to
Neo4j
v 1.0
neo4j

Overview

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

- Describe the components and benefits of the Neo4j Graph Platform.

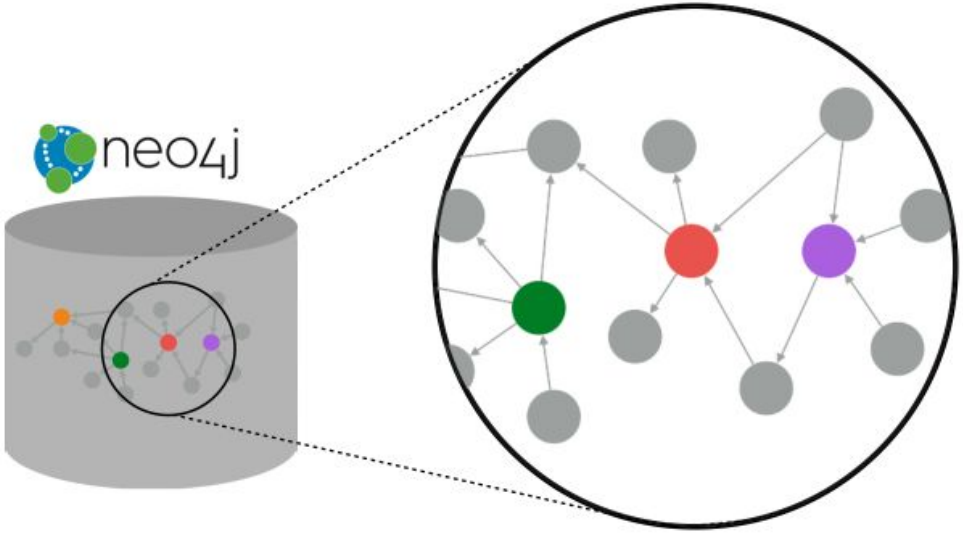
Neo4j Graph Platform

The Neo4j Graph Platform includes components that enable you to develop your graph-enabled application. To better understand the Neo4j Graph Platform, you will learn about these components and the benefits they provide.

The heart of the Neo4j Graph Platform is the Neo4j Database.

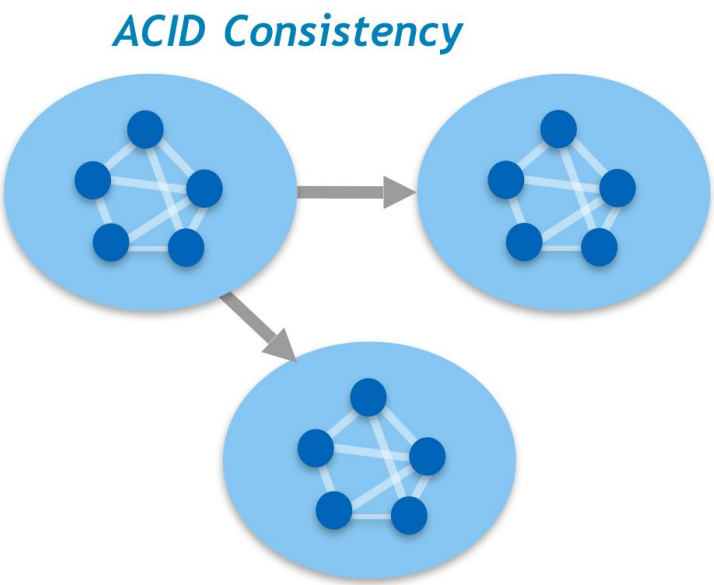
Neo4j Database: Index-free adjacency

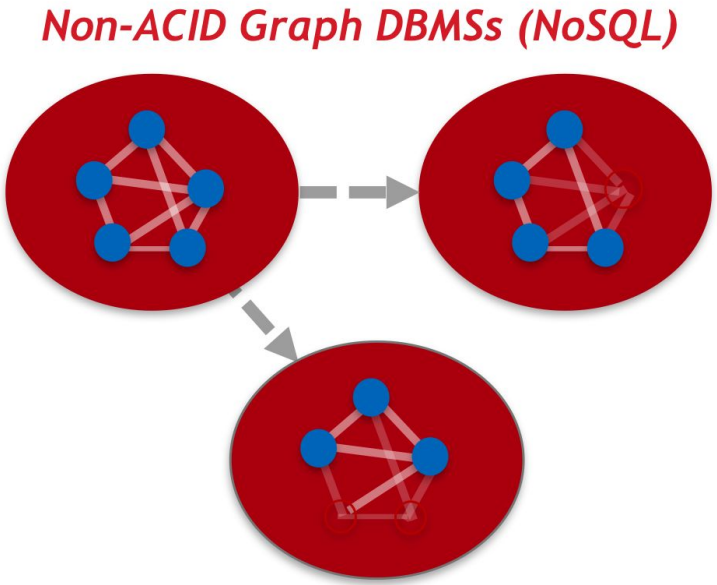
Nodes and relationships are stored on disk as a graph for fast navigational access using pointers.



Neo4j Database: ACID

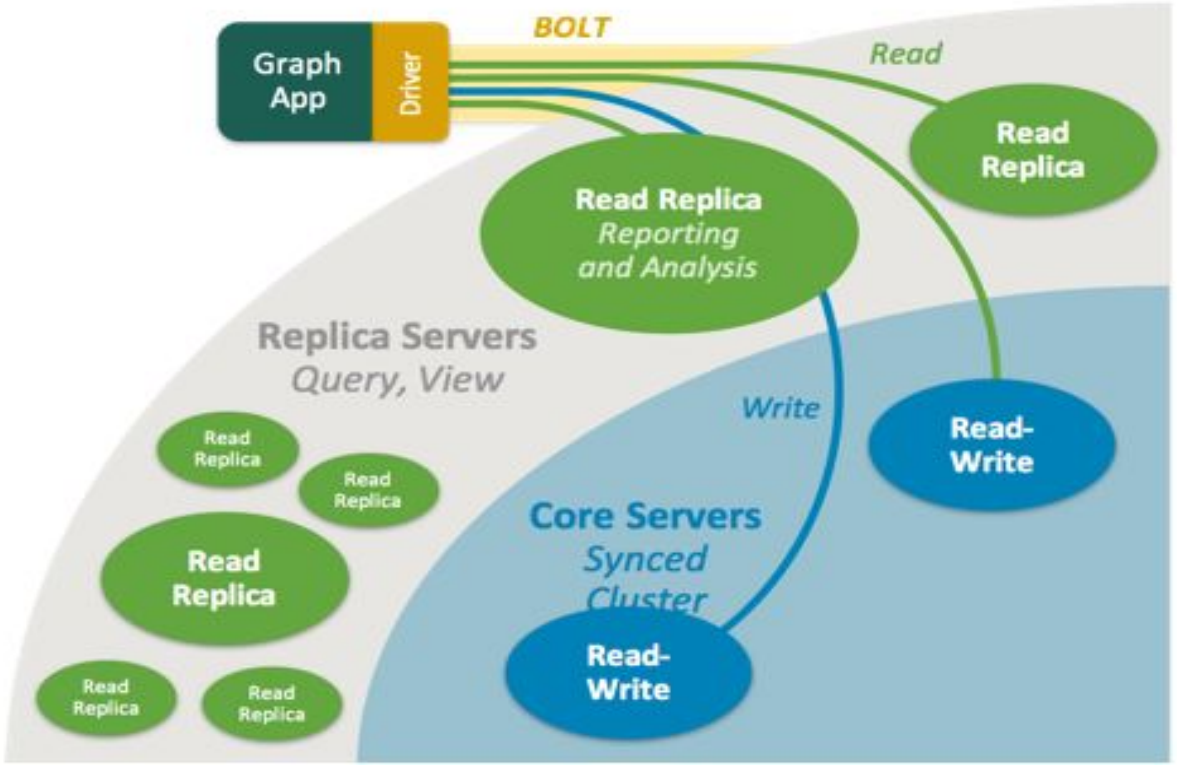
Transactional consistency - all updates either succeed or fail.





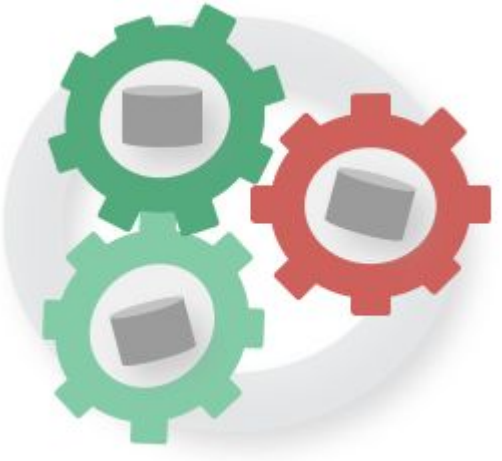
Clusters

ACID across locations.



Graph engine

- Interpret Cypher statements
- Store and retrieve data
- Kernel-level access to filesystem
- Scalable
- Performant



Language and driver support

- Cypher to access the database
- Open source Cypher
- You can write server-side extensions to access the database
- Out-of-the-box drivers to access the database via **bolt** protocol:
  - Java
  - JavaScript
  - Python
  - C#
  - Go
- Neo4j community contributions for other languages

Libraries

Out-of-the-box:

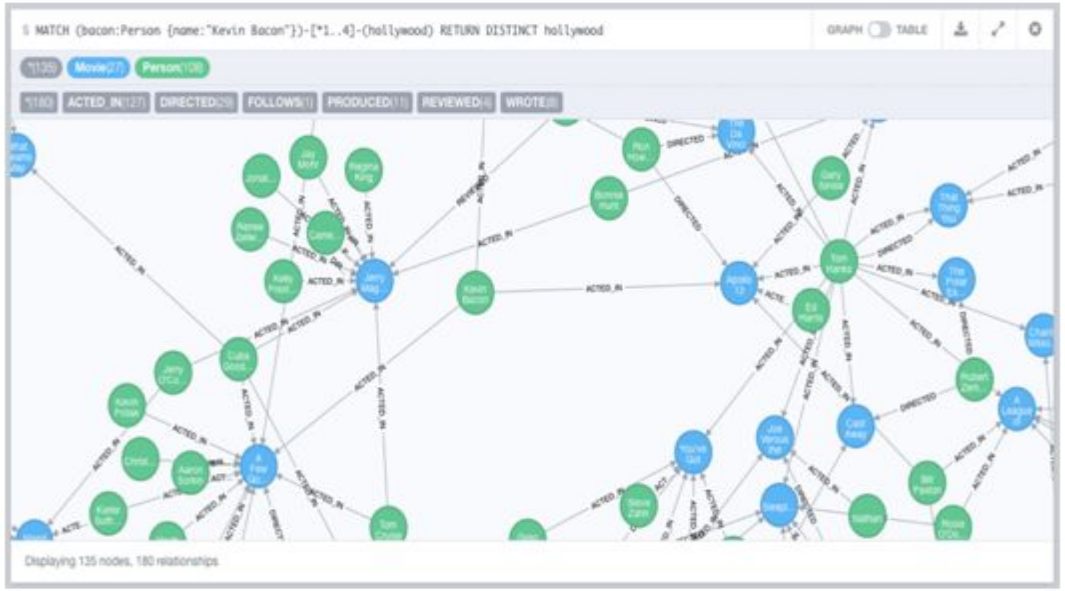
- Awesome Procedures on Cypher (APOC)
- Graph Algorithms
- GraphQL

Neo4j community has contributed many specialized libraries also.

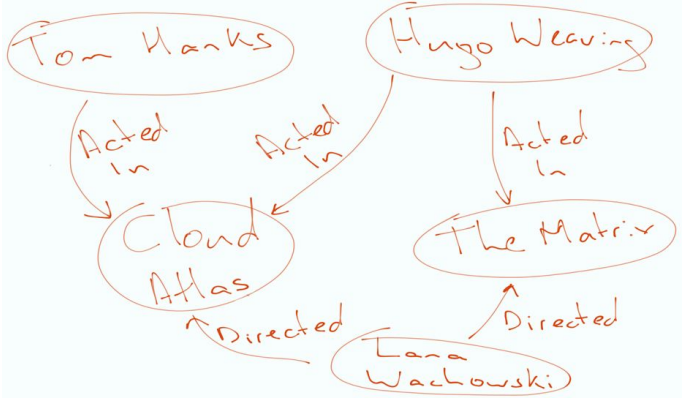
Tools

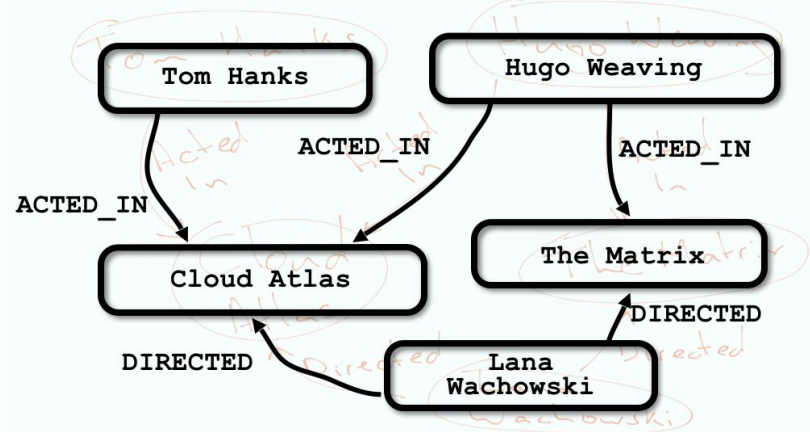
- Neo4j Desktop
- Neo4j Browser
- Neo4j Bloom
- Neo4j ETL Tool

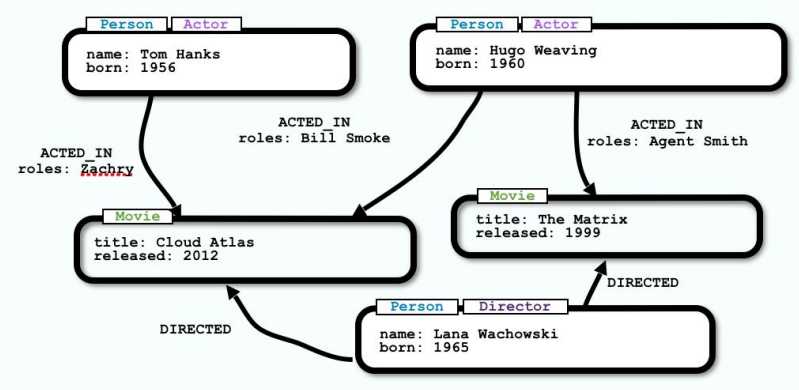
Neo4j community has contributed many specialized tools also.



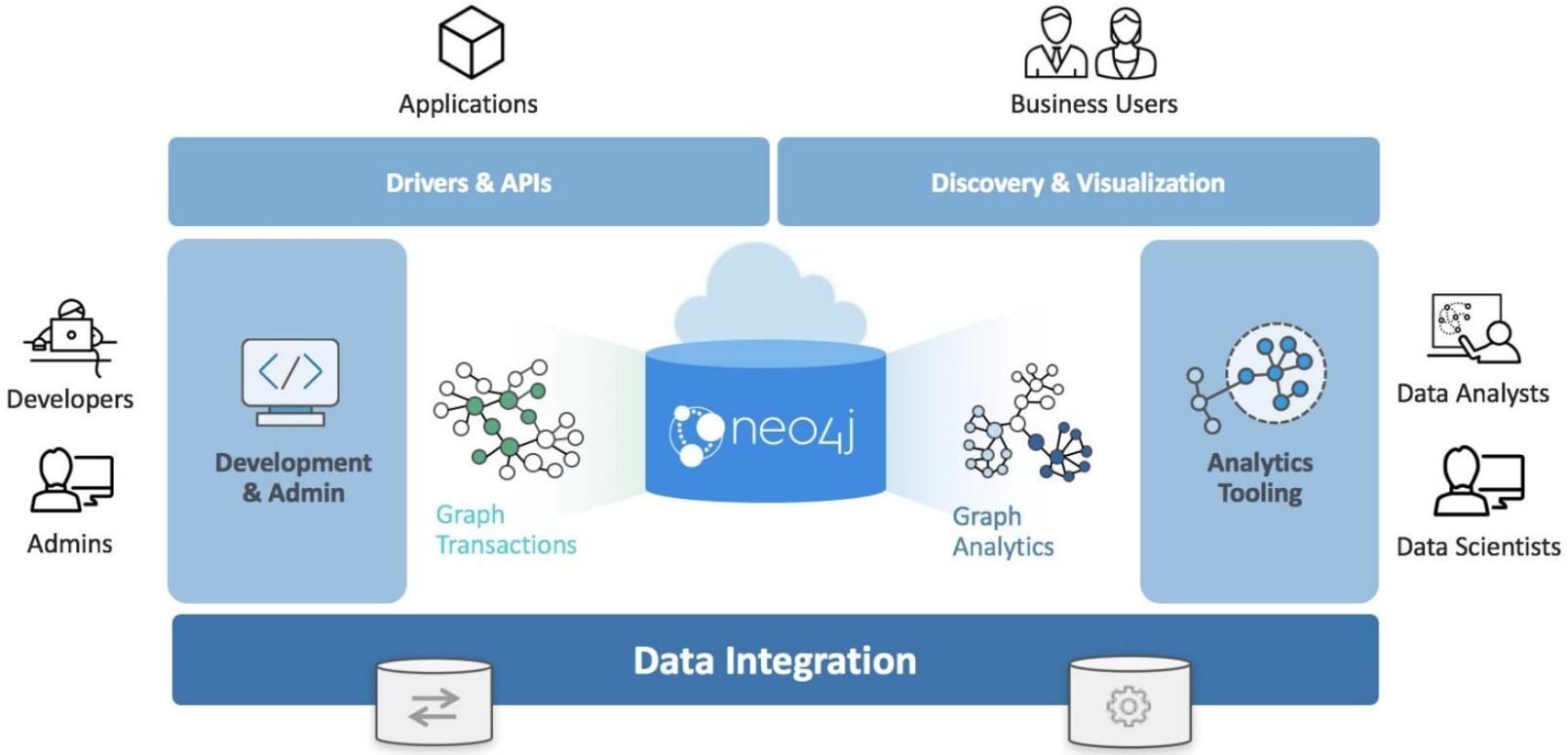
Whiteboard modeling







Neo4j Graph Platform architecture




Check your understanding

Question 1

What are some of the benefits provided by the Neo4j Graph Platform?

Select the correct answers.

- ❑ Database clustering
- ❑ ACID
- ❑ Index free adjacency
- ❑ Optimized graph engine

Answer 1

What are some of the benefits provided by the Neo4j Graph Platform?

Select the correct answers.

- ✓ Database clustering
- ✓ ACID
- ✓ Index free adjacency
- ✓ Optimized graph engine

Question 2

What libraries are included with Neo4j Graph Platform?

Select the correct answers.

- ❑ APOC
- ❑ JGraph
- ❑ GRAPH ALGORITHMS
- ❑ GraphQL

Answer 2

What libraries are included with Neo4j Graph Platform?

Select the correct answers.

- ✓ APOC
- ❑ JGraph
- ✓ GRAPH ALGORITHMS
- ✓ GraphQL

Question 3

What are some of the language drivers that come with Neo4j out of the box?

Select the correct answers.

- ❑ Java
- ❑ Ruby
- ❑ Python
- ❑ JavaScript

Answer 3

What are some of the language drivers that come with Neo4j out of the box?

Select the correct answers.

- ✓ Java
- ❑ Ruby
- ✓ Python
- ✓ JavaScript

Summary

You should be able to:

- Describe the components and benefits of the Neo4j Graph Platform.


Setting Up Your
Development Environment
v 1.0
neo4j

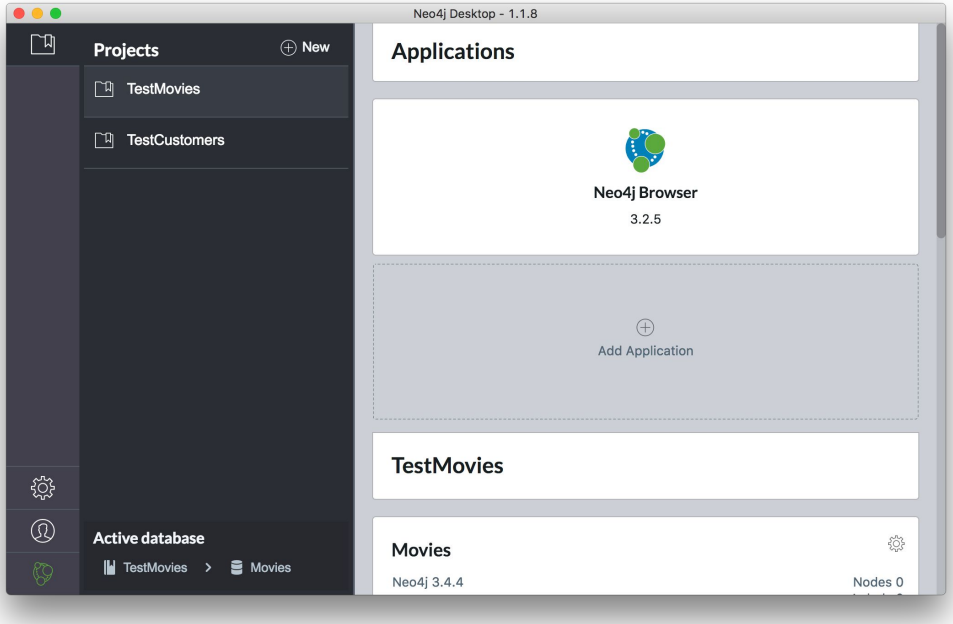
Overview

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

- Determine the development environment that is best for you:
  - Install and start using the Neo4j Desktop.
  - Create a Neo4j Sandbox for learning Neo4j.
- Start using Neo4j Browser.

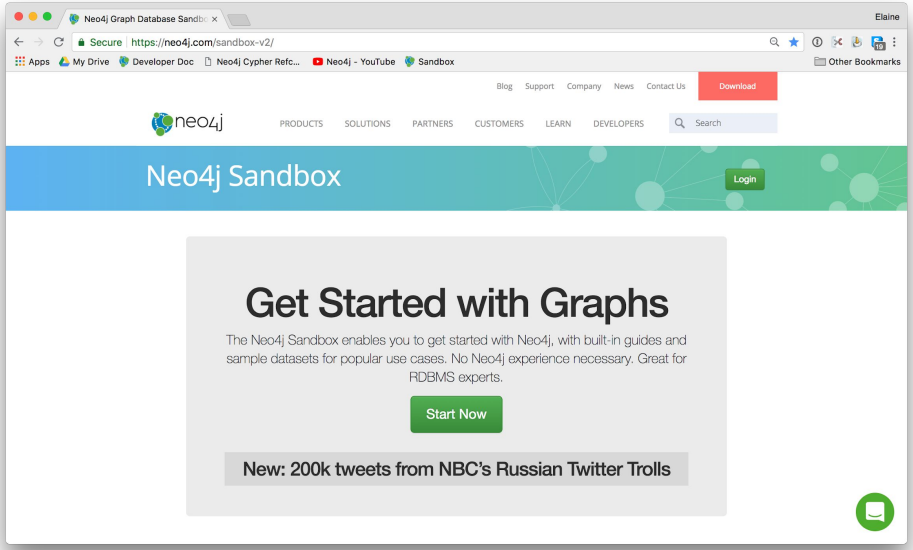
Neo4j Desktop

- Create local databases
- Manage multiple projects
- Manage Database Server
- Start Neo4j Browser instances
- Install plugins (libraries) for use with a project
- OS X, Linux, Windows



Neo4j Sandbox

- Web browser access to Neo4j Database Server and Neo4j Database in the cloud
- Comes with a blank or pre-populated database
- Temporary access
- Save Cypher scripts for use in other sandboxes or Neo4j projects
- Instance lives for up to ten days
- No need to install Neo4j on your machine



neo4j

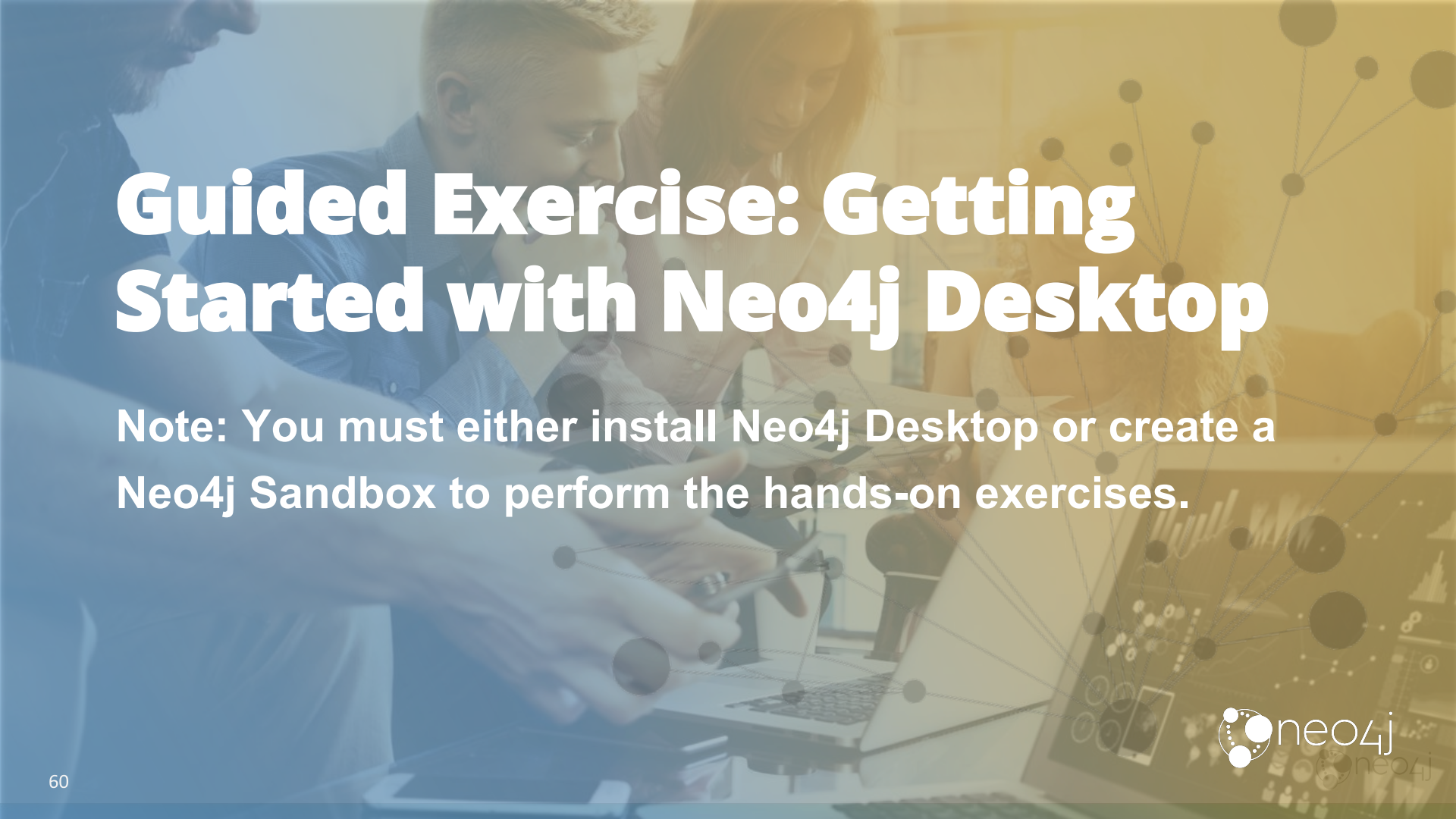
Setting up your development environment

If using Neo4j Sandbox:

1. Start a Neo4j Sandbox (use latest Neo4j release).
   a. Has a blank database that is started.
2. Click the link to access Neo4j Browser.

If using Neo4j Desktop:

1. Install Neo4j Desktop.
2. In a project, create a local graph (database).
3. Start the database.
4. Click the Neo4j Browser application.



Guided Exercise: Getting Started with Neo4j Desktop

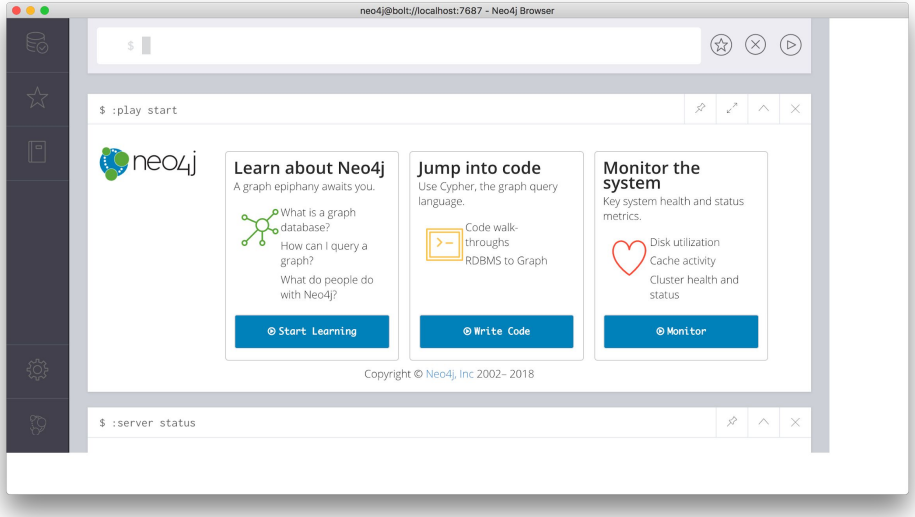
Note: You must either install Neo4j Desktop or create a Neo4j Sandbox to perform the hands-on exercises.

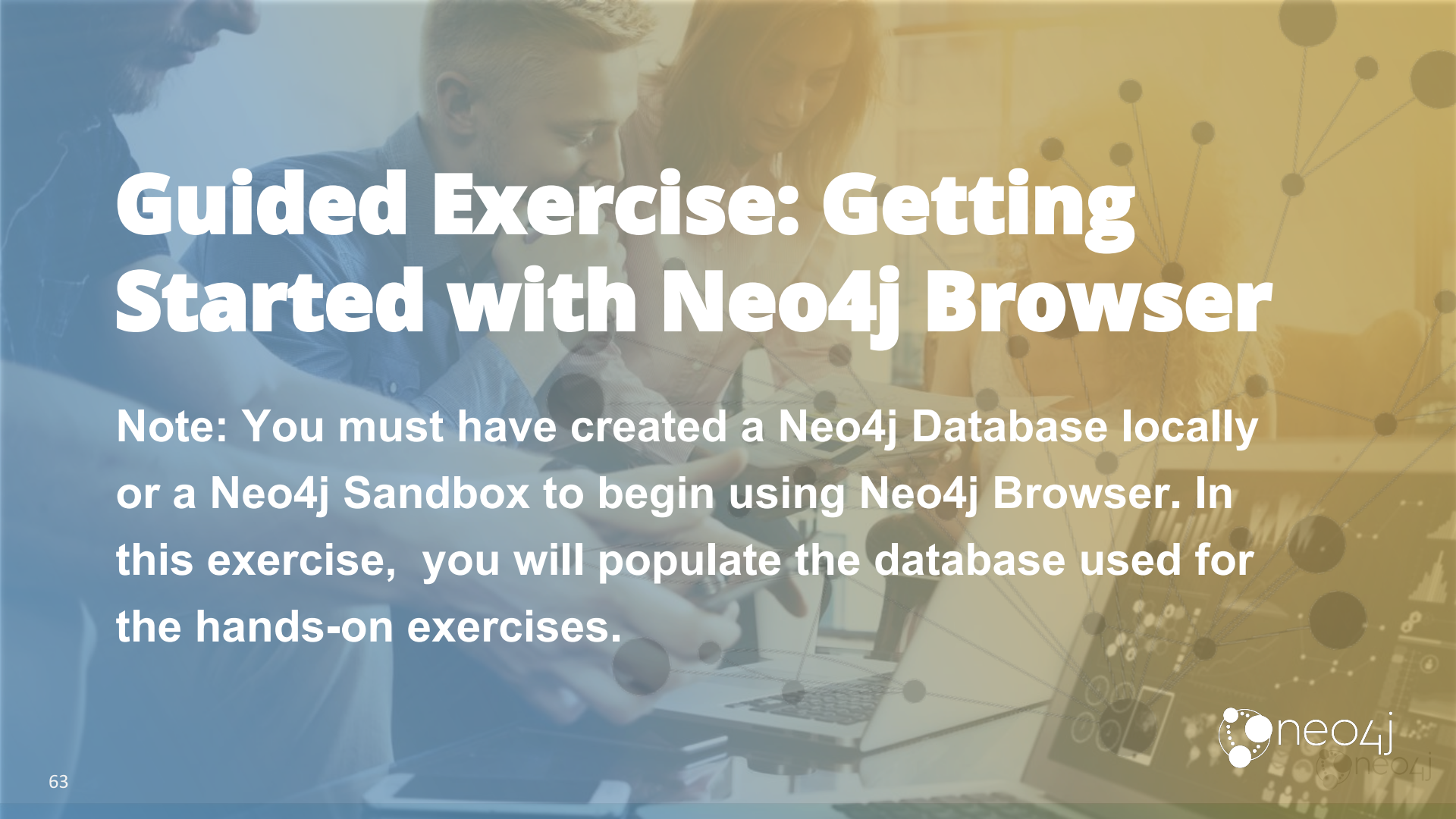
Guided Exercise: Creating a Neo4j Sandbox

Note: You must either install Neo4j Desktop or create a Neo4j Sandbox to perform the hands-on exercises.

Neo4j Browser

- Web browser access to Neo4j Database Server and Neo4j Database
- Access local database (Neo4j Desktop) or database in the cloud (Sandbox)
- Access the database with commands or Cypher statements





Guided Exercise: Getting Started with Neo4j Browser

Note: You must have created a Neo4j Database locally or a Neo4j Sandbox to begin using Neo4j Browser. In this exercise, you will populate the database used for the hands-on exercises.

Check your understanding

Question 1

What development environment should you use if you want to develop a graph-enabled application using a local Neo4j Database?

Select the correct answer.

- ❏ Neo4j Desktop
- ❏ Neo4j Sandbox

Answer 1

What development environment should you use if you want to develop a graph-enabled application using a local Neo4j Database?

Select the correct answer.

- ✓ Neo4j Desktop
- ❑ Neo4j Sandbox

Question 2

What development environment should you use if you want develop a graph-enabled application using a temporary, cloud-based Neo4j Database?

Select the correct answer.

- ❏ Neo4j Desktop
- ❏ Neo4j Sandbox

Answer 2

What development environment should you use if you want develop a graph-enabled application using a temporary, cloud-based Neo4j Database?

Select the correct answer.

❏ Neo4j Desktop

✓ Neo4j Sandbox

Question 3

Which Neo4j Browser command do you use to view a browser guide for the Movie graph?

Select the correct answer.

- ❏ MATCH (Movie Graph)
- ❏ :MATCH (Movie Graph)
- ❏ play Movie Graph
- ❏ :play Movie Graph

Answer 3

Which Neo4j Browser command do you use to view a browser guide for the Movie graph?

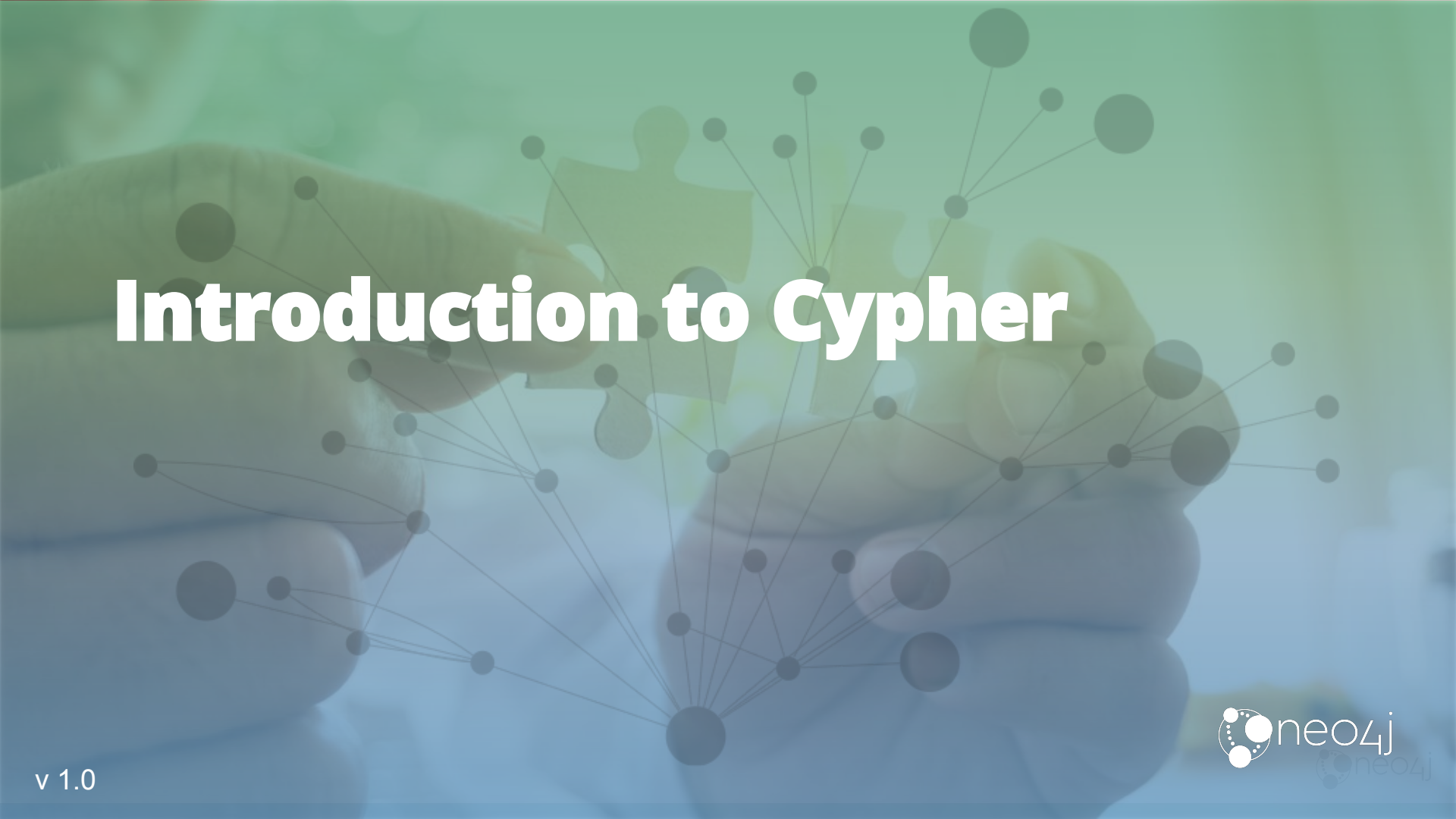
Select the correct answer.

- ❏ MATCH (Movie Graph)
- ❏ :MATCH (Movie Graph)
- ❏ play Movie Graph
- ✓ :play Movie Graph

Summary

You should be able to:

- Determine the development environment that is best for you:
  - Install and start using the Neo4j Desktop.
  - Create a Neo4j Sandbox for learning Neo4j.
- Start using Neo4j Browser.


Introduction to Cypher
neo4j
v 1.0

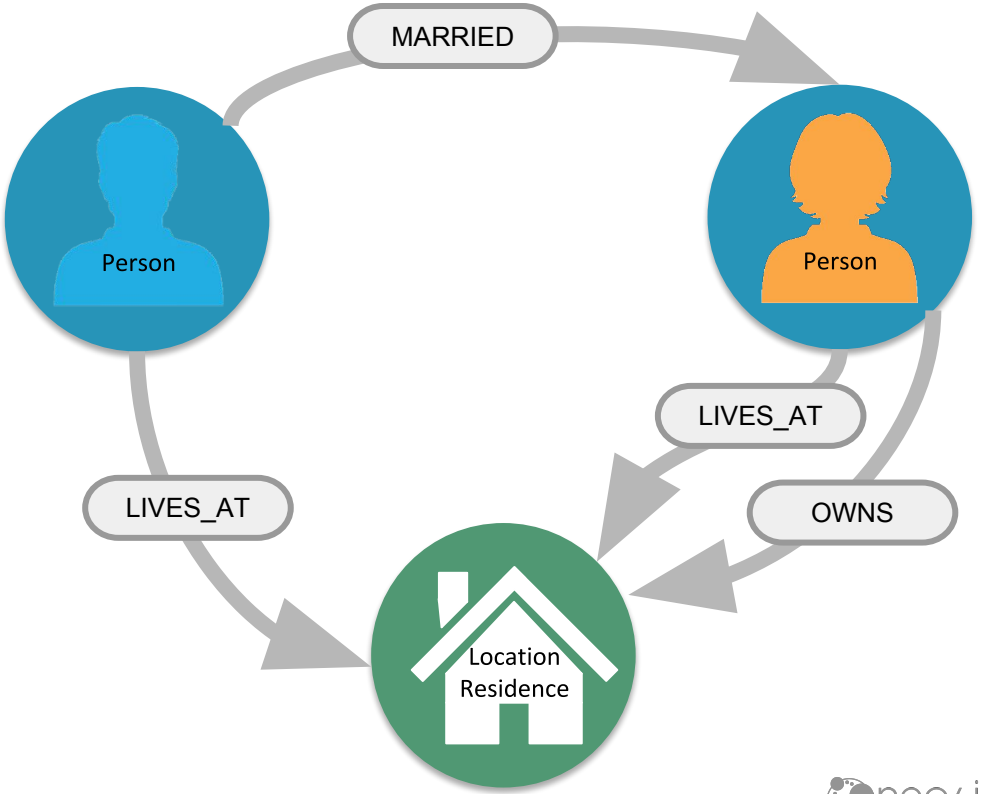
Overview

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

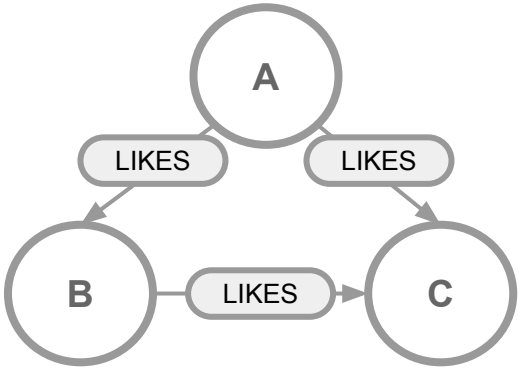
- Retrieve nodes from the graph.
- Filter nodes retrieved using labels and property values of nodes.
- Retrieve property values from nodes in the graph.
- Filter nodes retrieved using relationships.

What is Cypher?

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

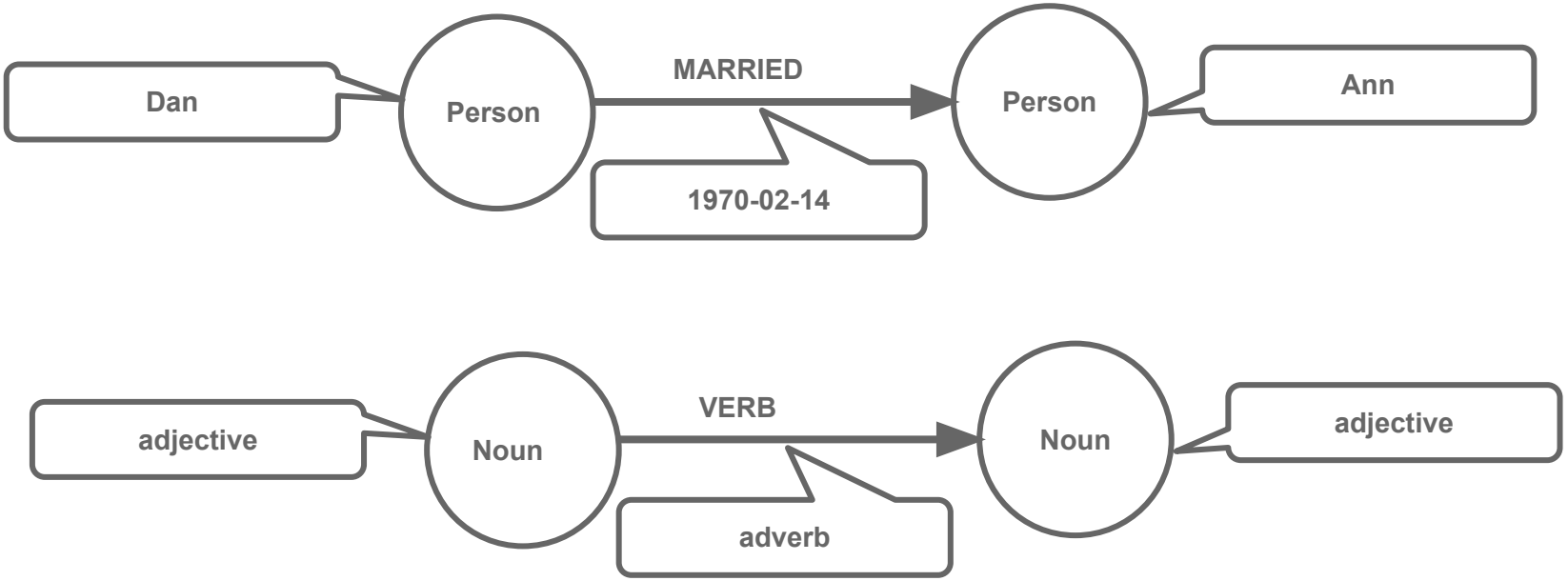


Cypher is ASCII Art



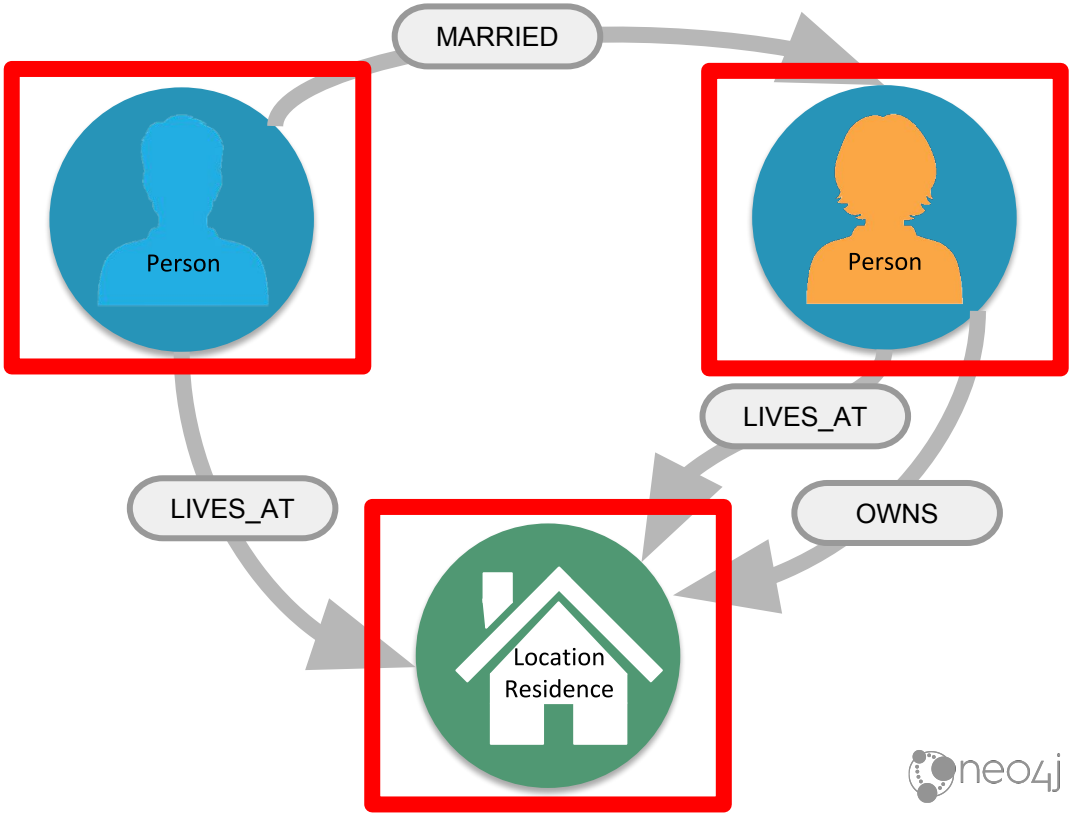

```
(A)-[:LIKES]->(B),(A)-[:LIKES]->(C),(B)-[:LIKES]->(C)

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

Cypher is readable



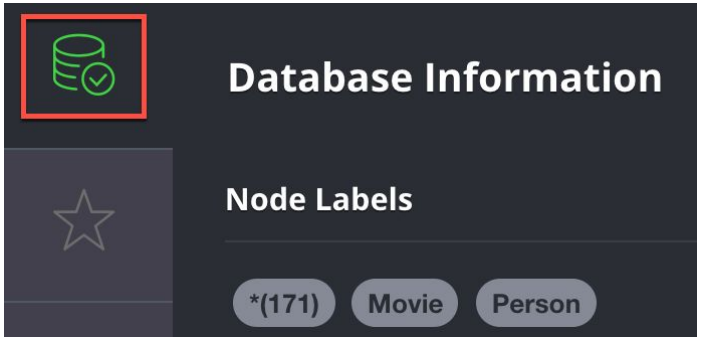
Nodes

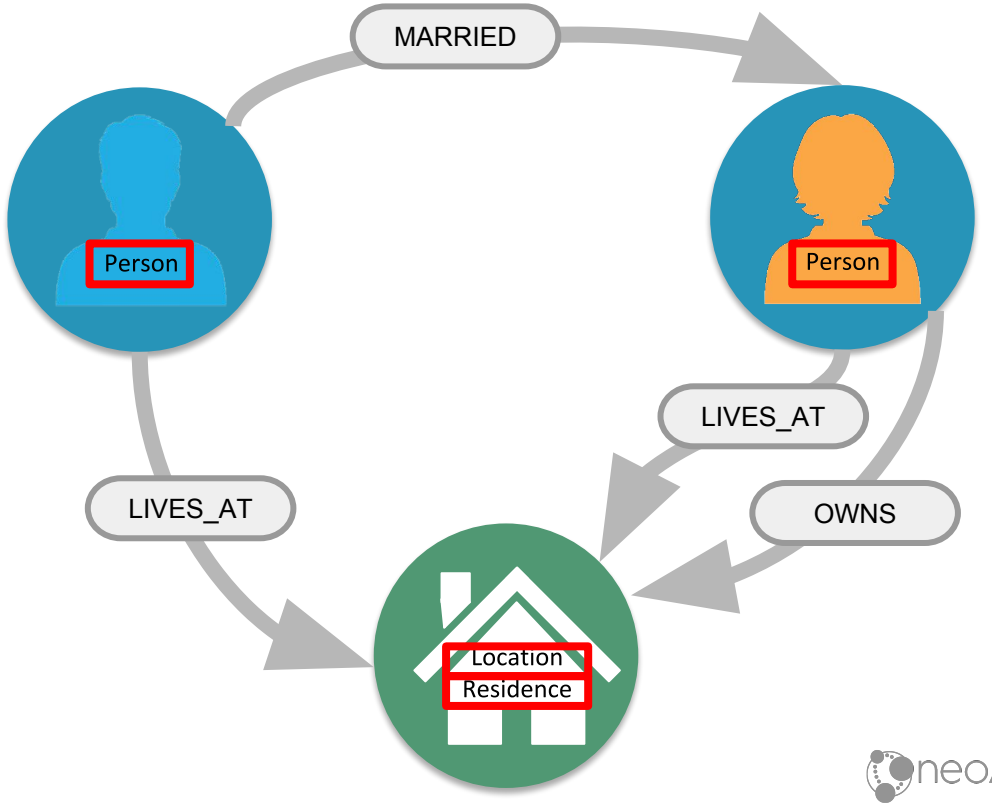
```
()
(p)
(l)
(n)
```



Labels

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



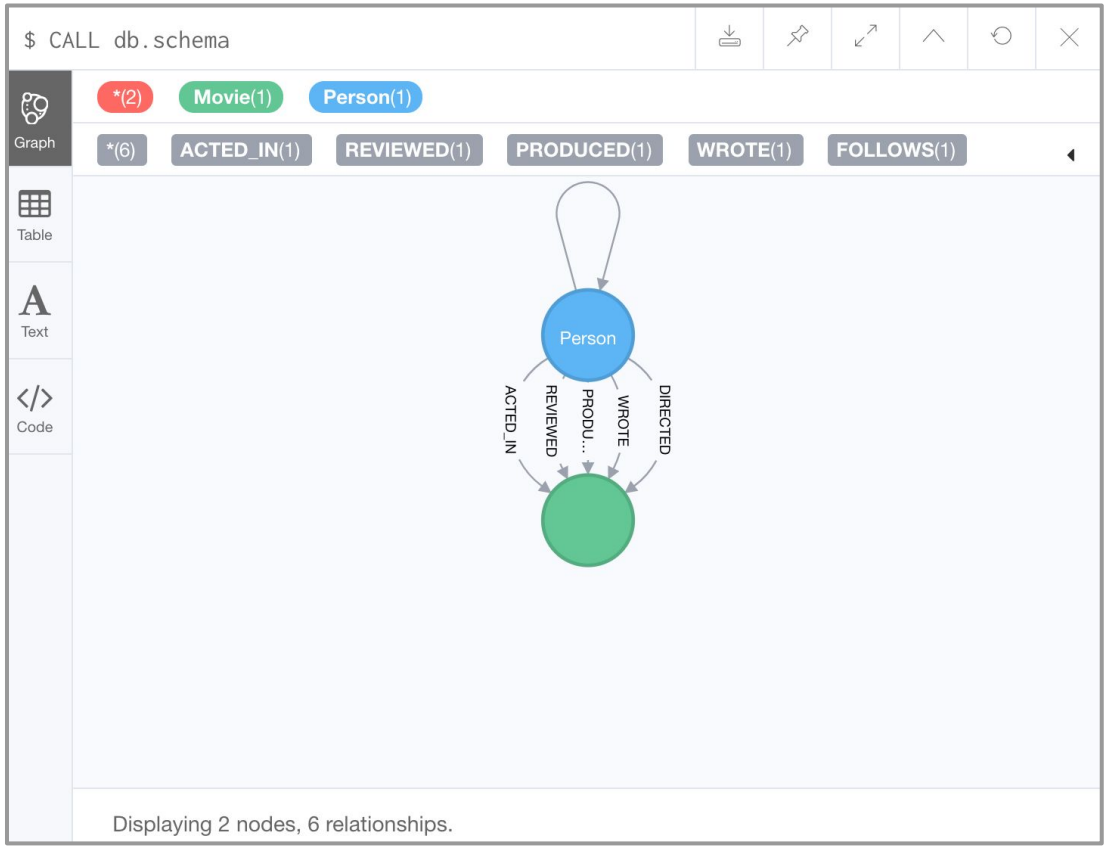


neo4j

Comments in Cypher

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

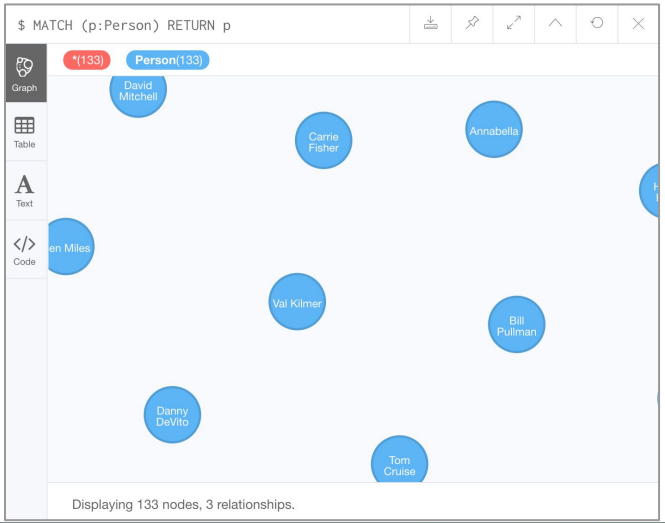
Examining the data model



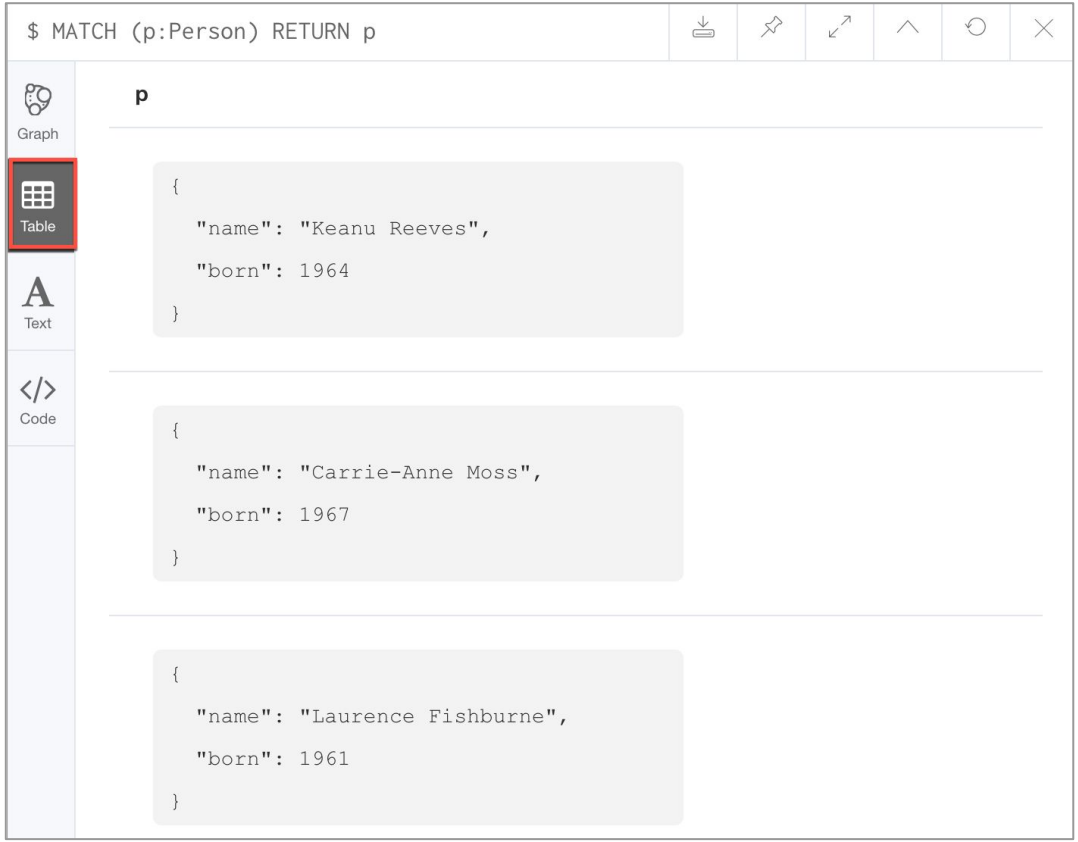
Using MATCH to retrieve nodes

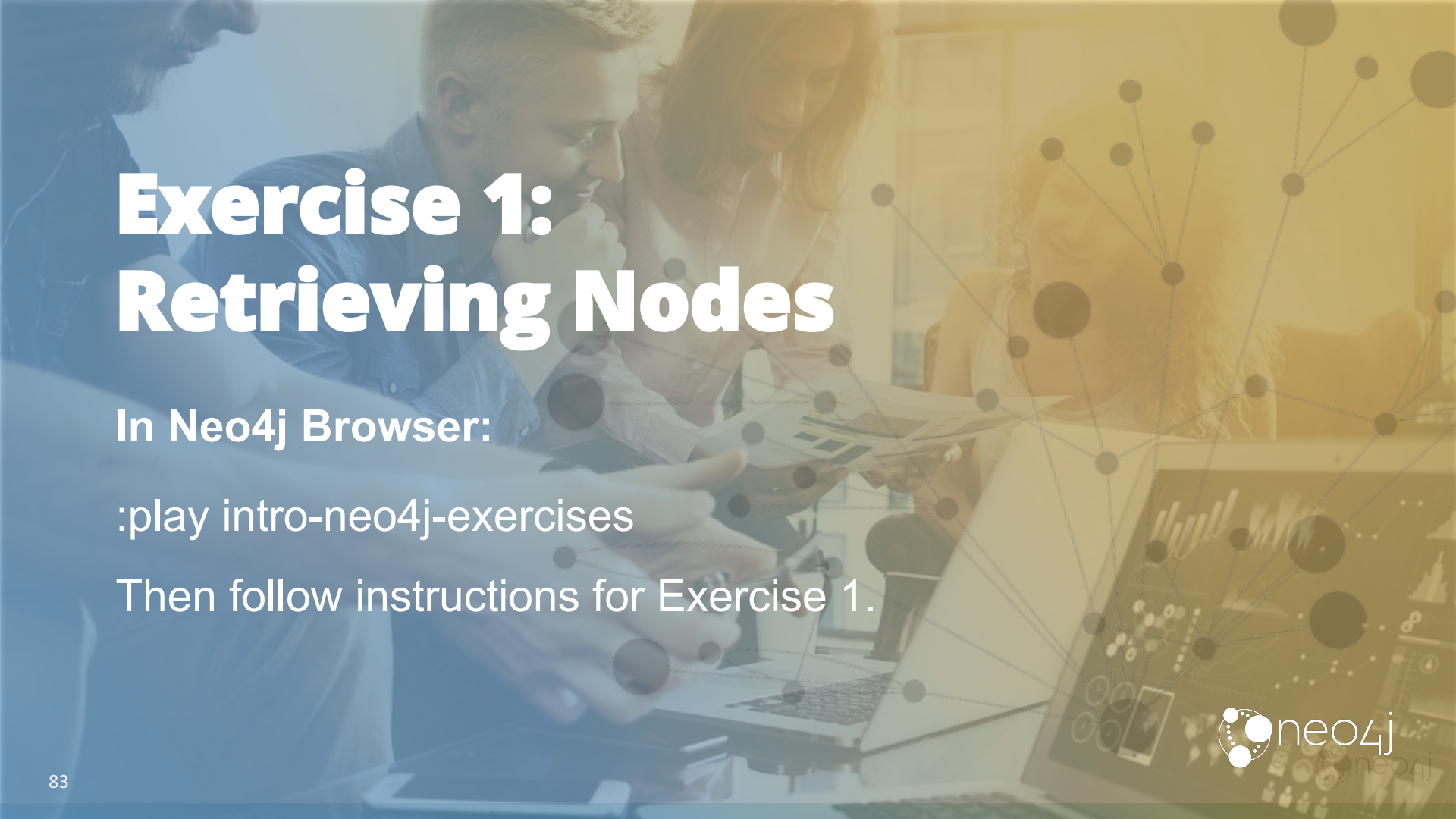
```
MATCH (n) // returns all nodes in the graph
RETURN n
```

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



Viewing nodes as table data





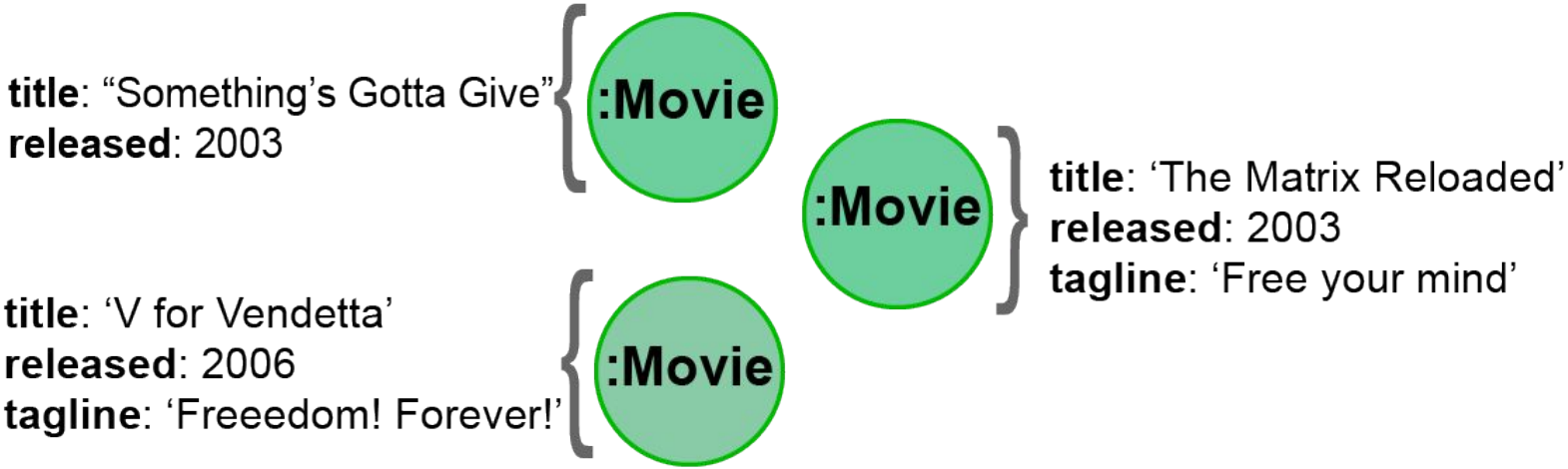
Exercise 1: Retrieving Nodes

In Neo4j Browser:

:play intro-neo4j-exercises

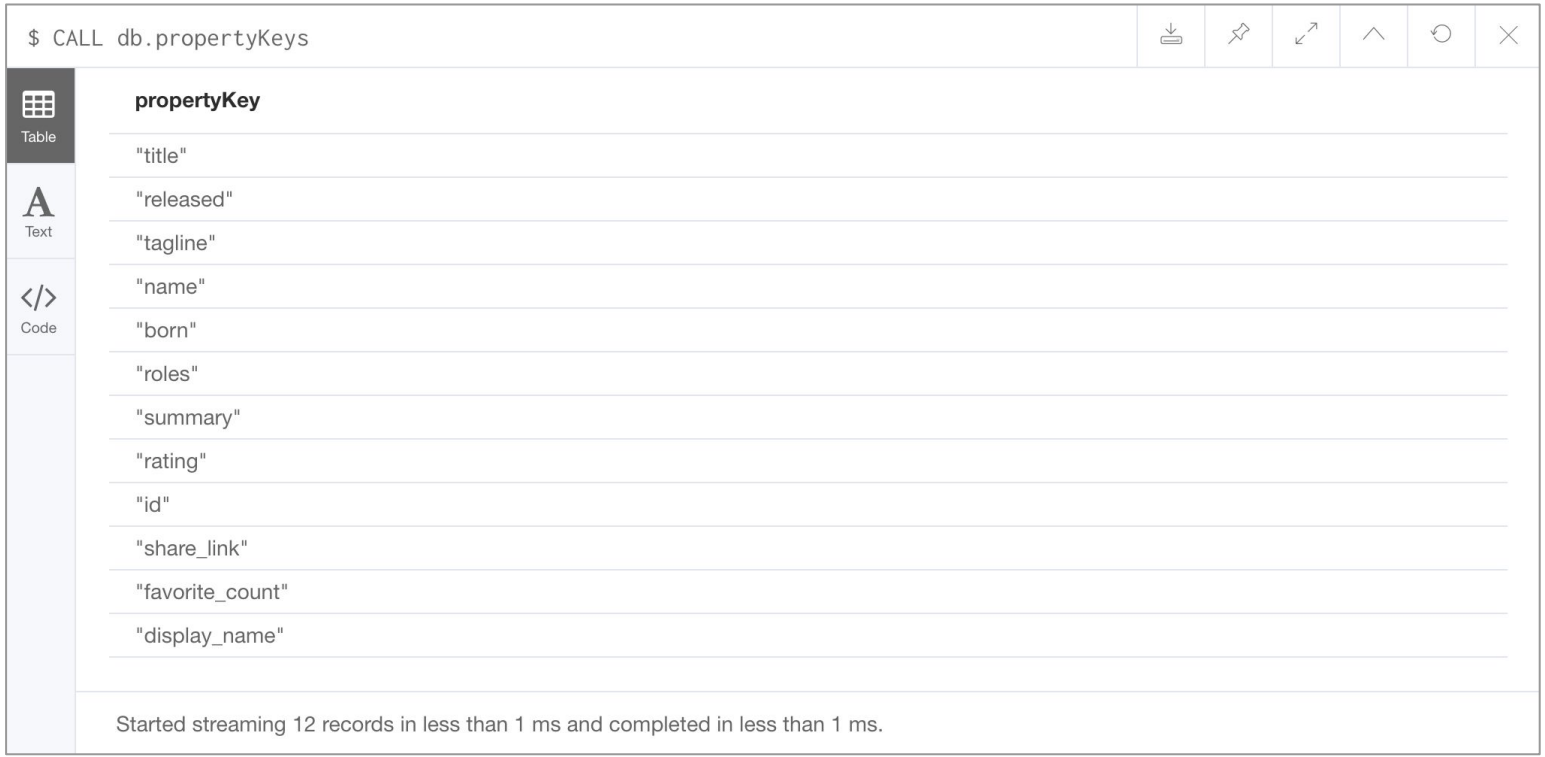
Then follow instructions for Exercise 1.

Properties


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

Examining property keys

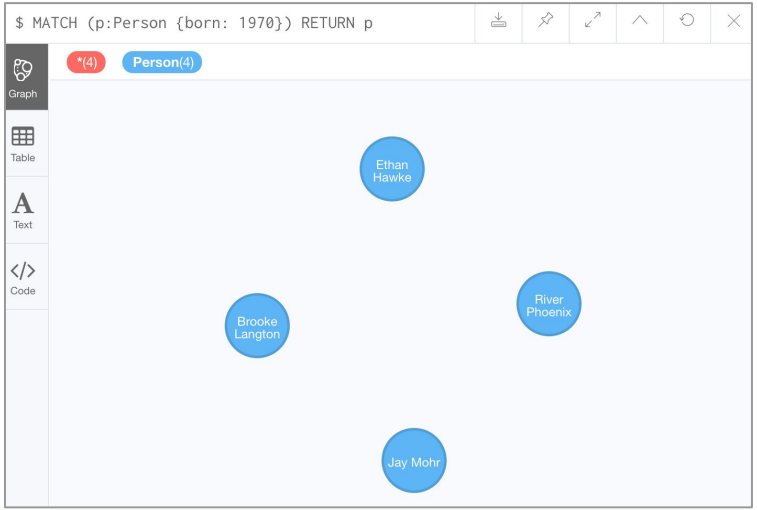
```
CALL db.propertyKeys
```



Retrieving nodes filtered by a property value - 1

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

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



Retrieving nodes filtered by a property value - 2

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

Free your mind, returning the nodes:

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

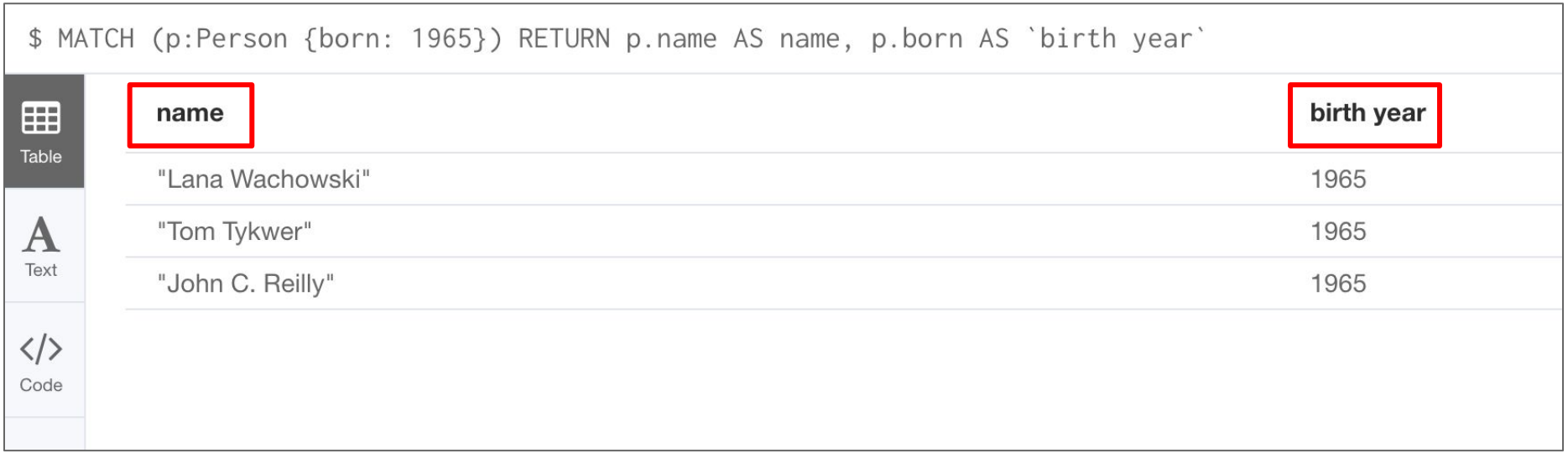
Returning property values

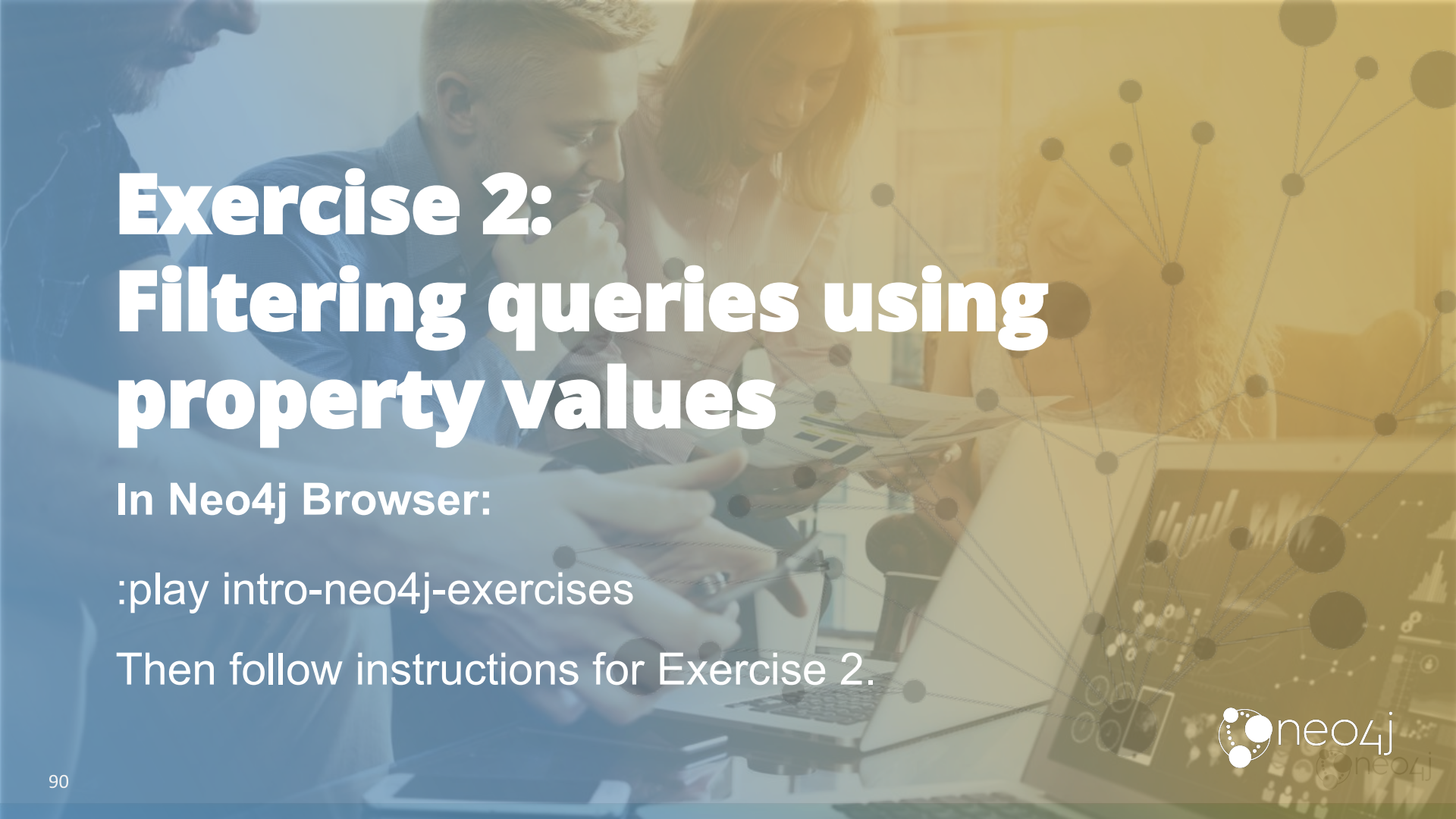
Find all people born in *1965* and return their names:

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

Specifying aliases

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





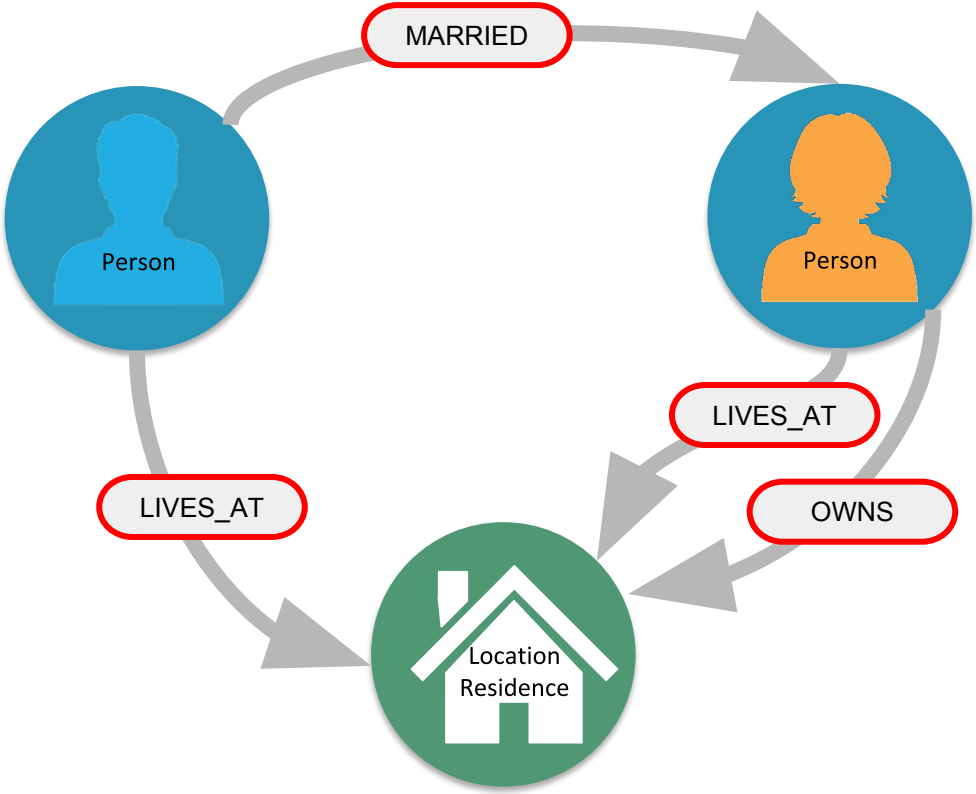
Exercise 2: Filtering queries using property values

In Neo4j Browser:

:play intro-neo4j-exercises

Then follow instructions for Exercise 2.

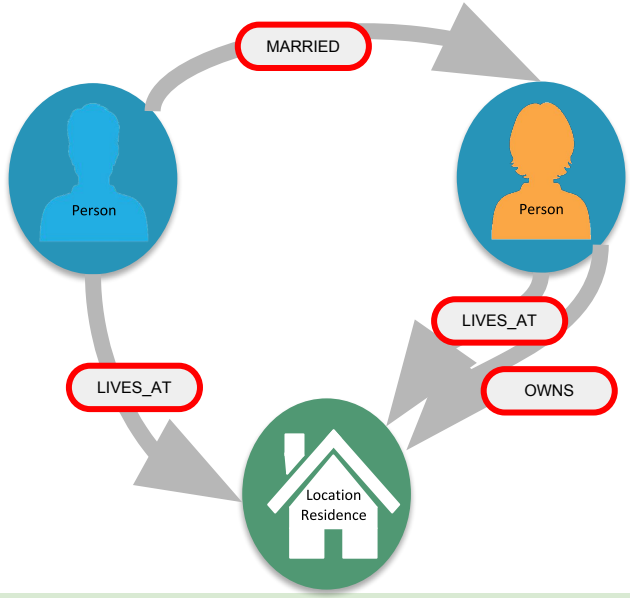
Relationships



ASCII art for nodes and relationships

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

Querying using relationships

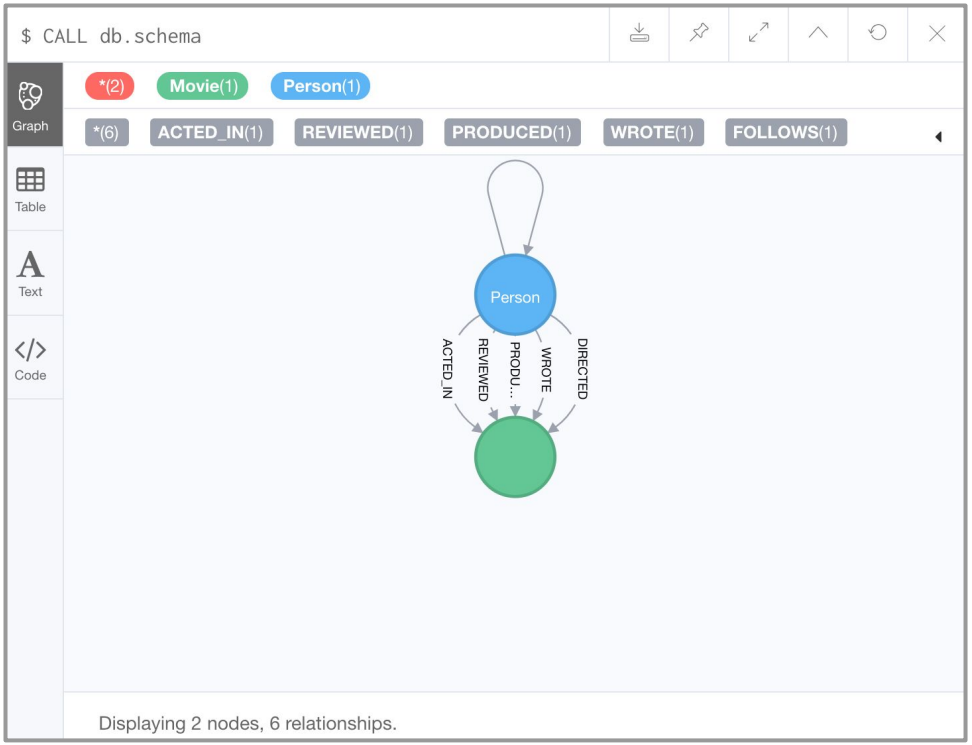



```
MATCH (p:Person)-[:LIVES_AT]->(h:Residence)
RETURN p.name, h.address
```

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

Examining relationships

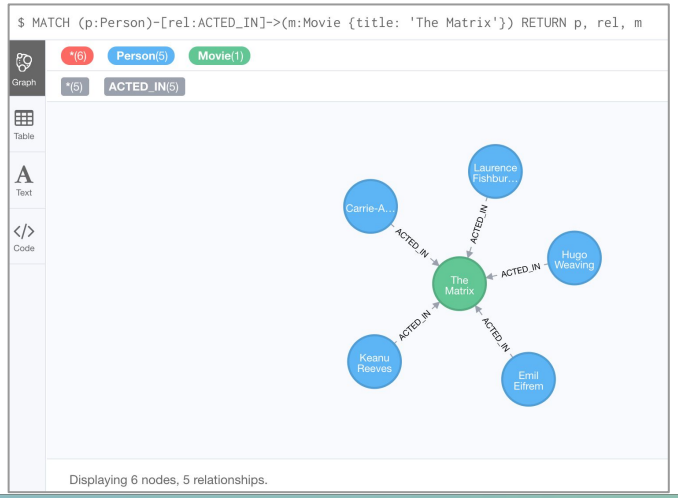
```
CALL db.schema
```



Using a relationship in a query

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

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



Querying by multiple relationships

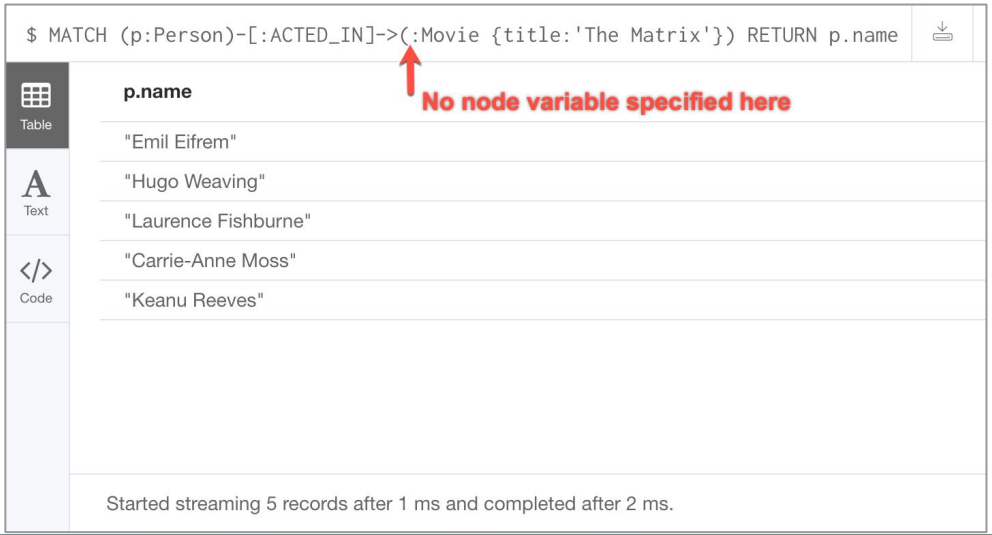
Find all movies that *Tom Hanks* acted in or directed and return the title of the move:

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

Using anonymous nodes in a query

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

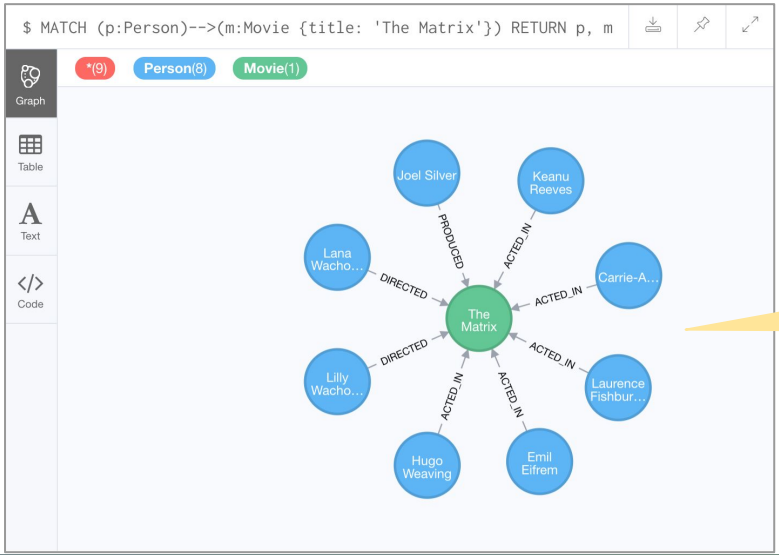
```
MATCH (p:Person)-[:ACTED_IN]->(:Movie {title: 'The Matrix'})
RETURN p.name
```



Using an anonymous relationship for a query

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

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



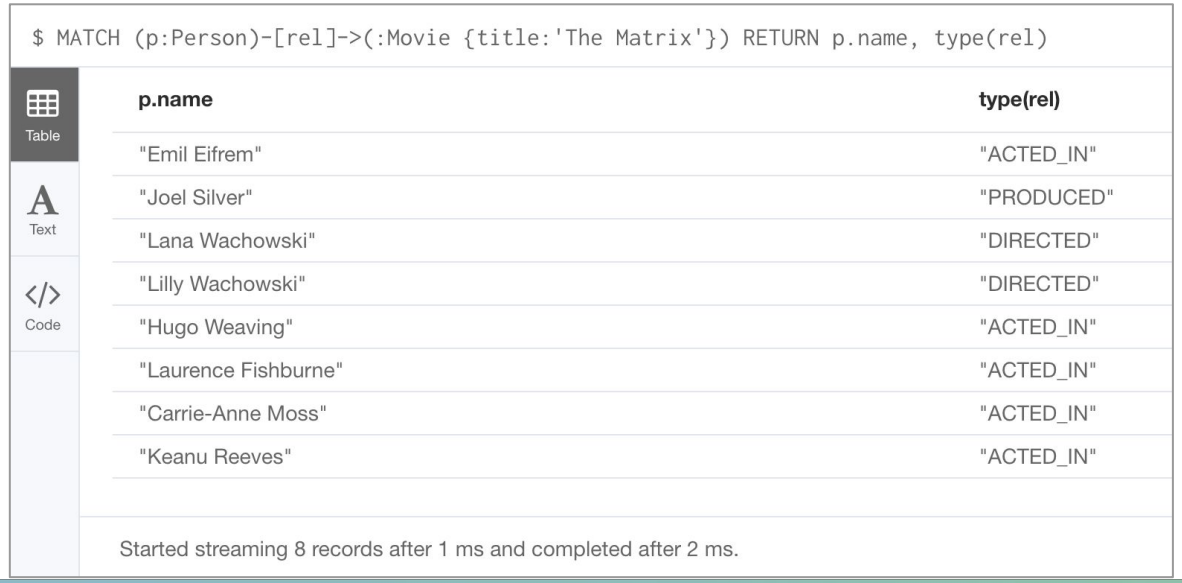
Connect result nodes enabled in Neo4j Browser

neo4j

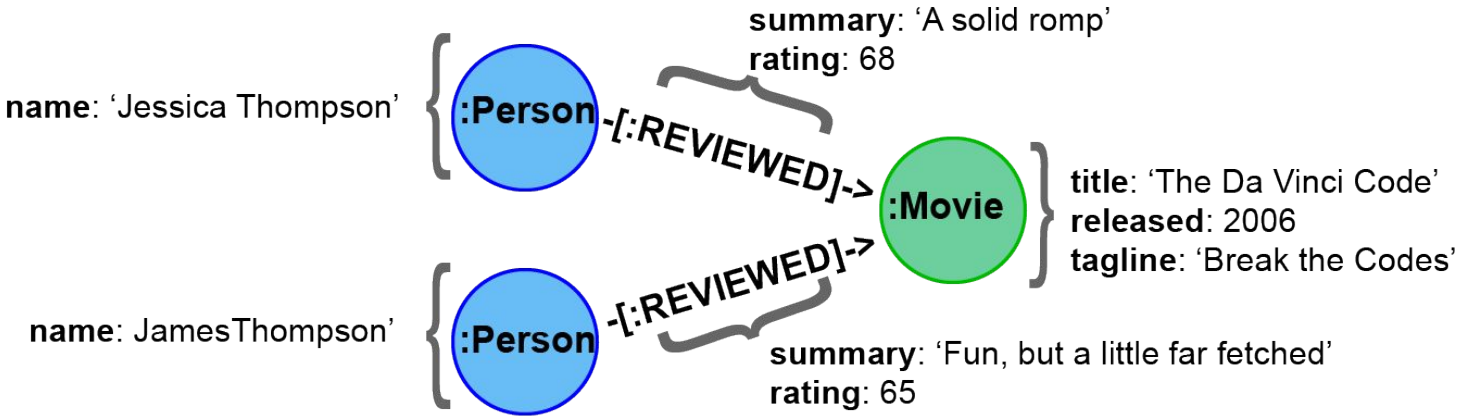
Retrieving relationship types

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

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



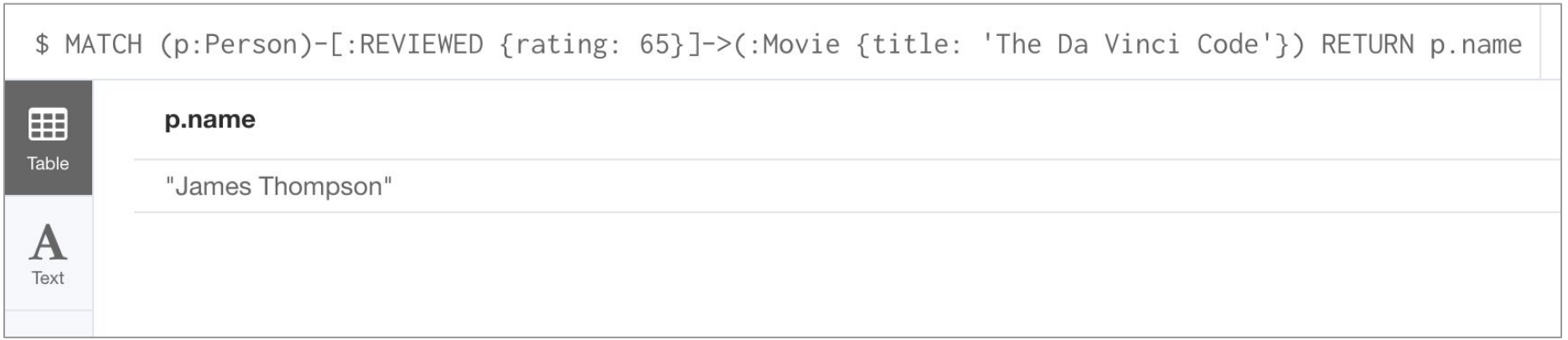
Retrieving properties for a relationship - 1



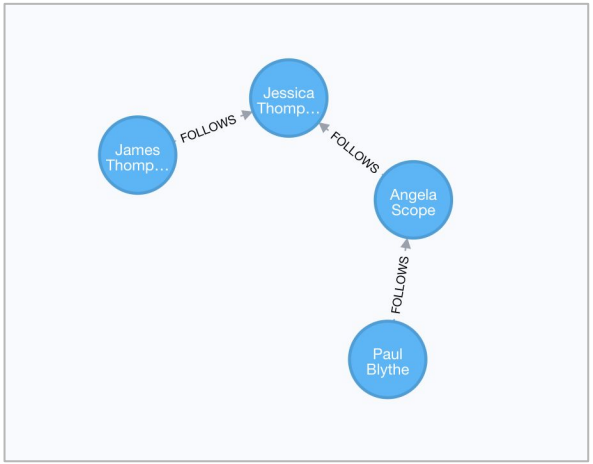
Retrieving properties for a relationship - 2

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

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

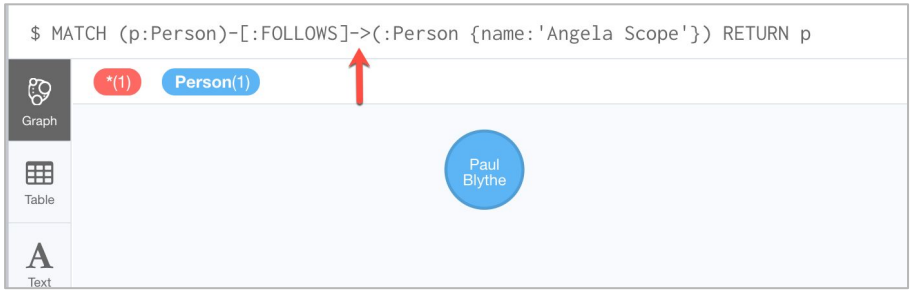


Using patterns for queries - 1

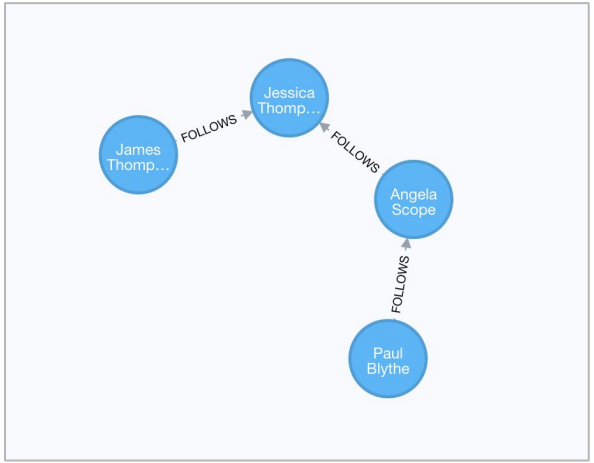


Find all people who follow *Angela Scope*, returning the nodes:

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

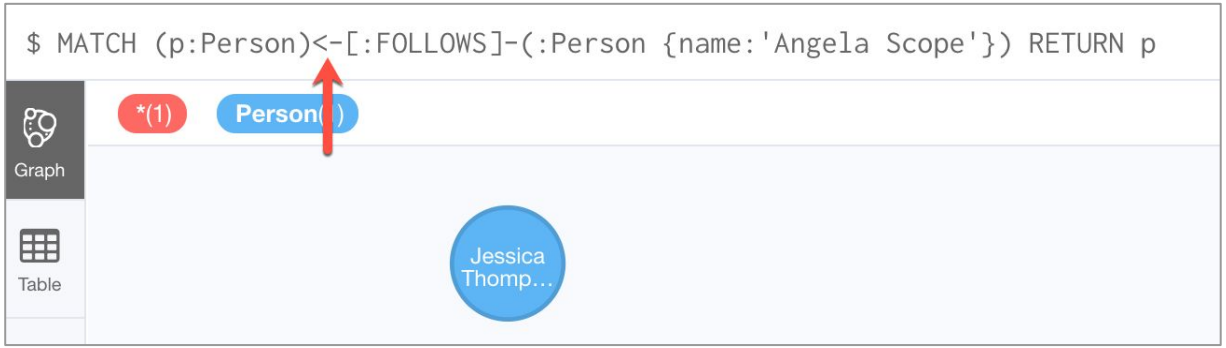


Using patterns for queries - 2

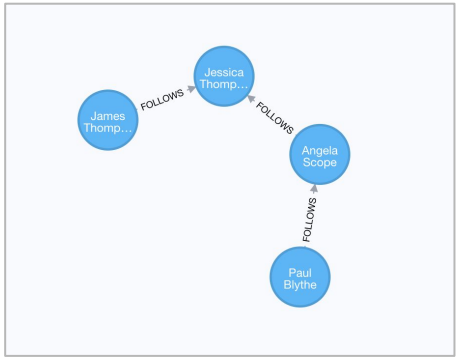


Find all people who *Angela Scope* follows, returning the nodes:

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

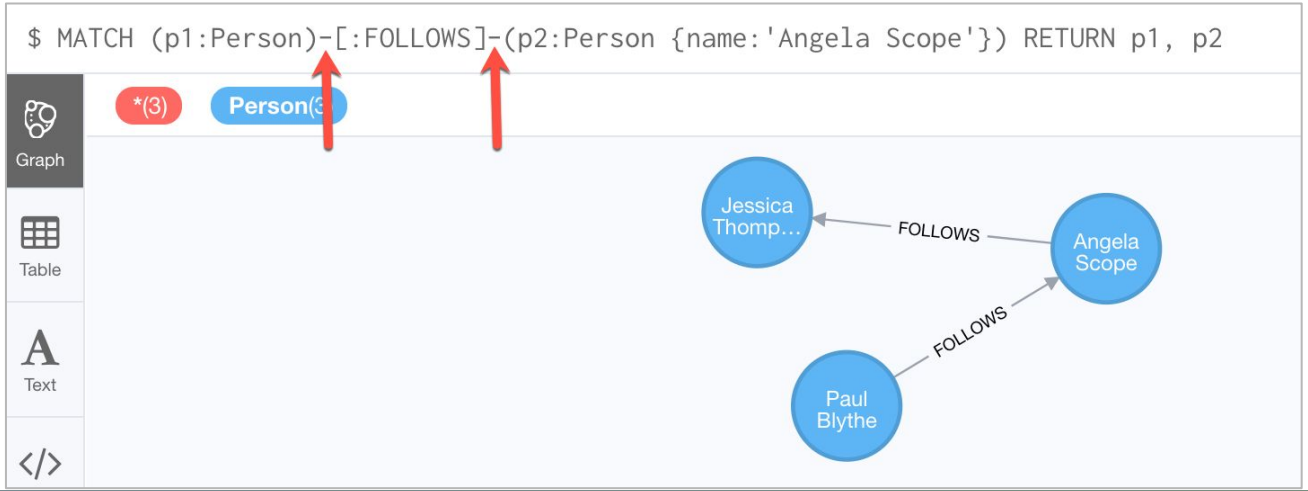


Querying by any direction of the relationship



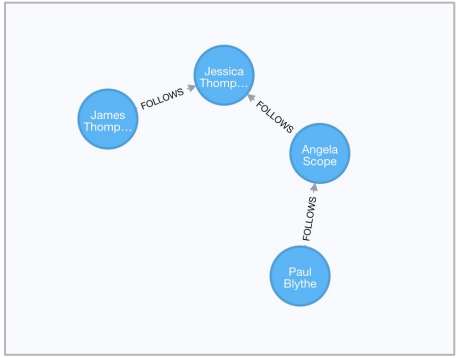
Find all people who follow *Angela Scope* or who *Angela Scope* follows, returning the nodes:

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



neo4j

Traversing relationships - 1

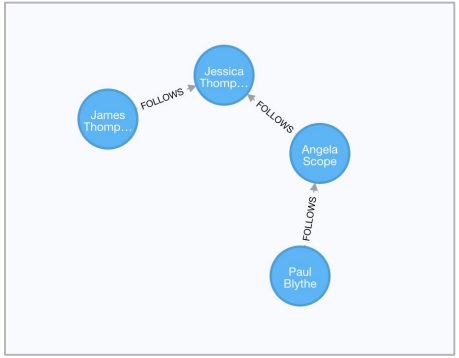


Find all people who follow anybody who follows *Jessica Thompson* returning the people as nodes:

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

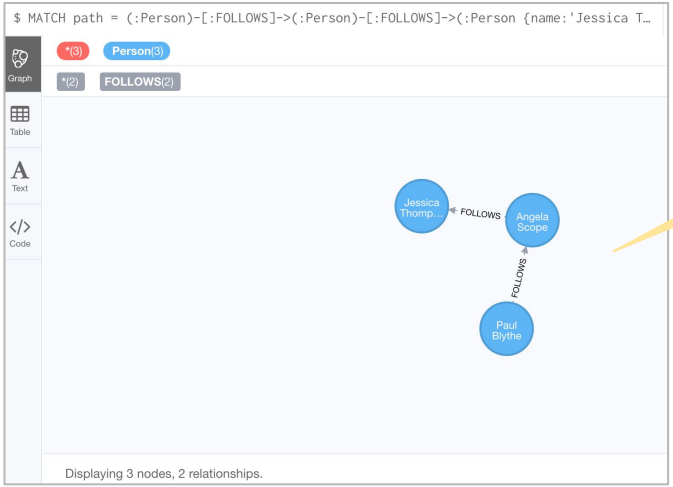
neo4j

Traversing relationships - 2



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

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



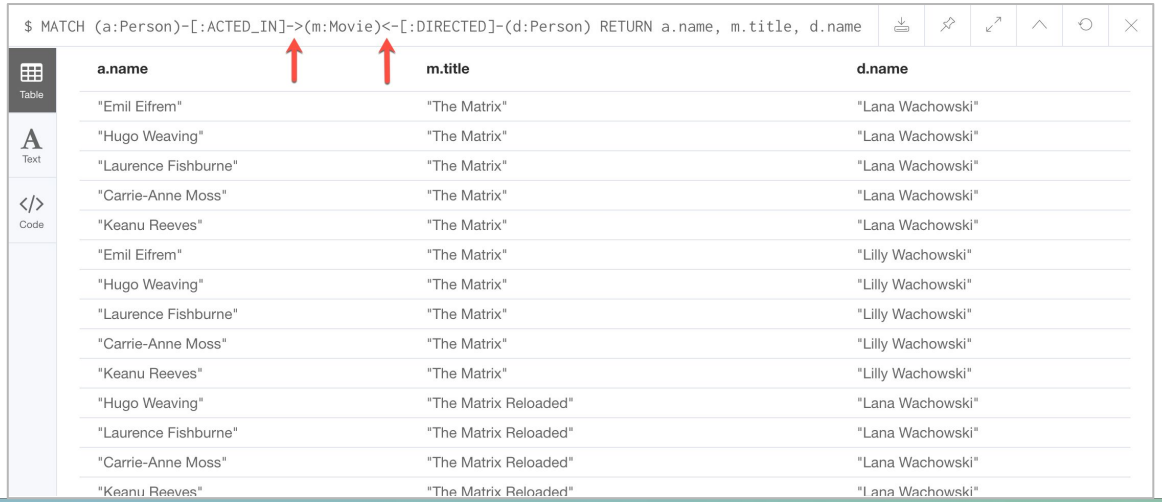
Sub-graph

neo4j

Using relationship direction to optimize a query

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

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



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

Cypher style recommendations - 1

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

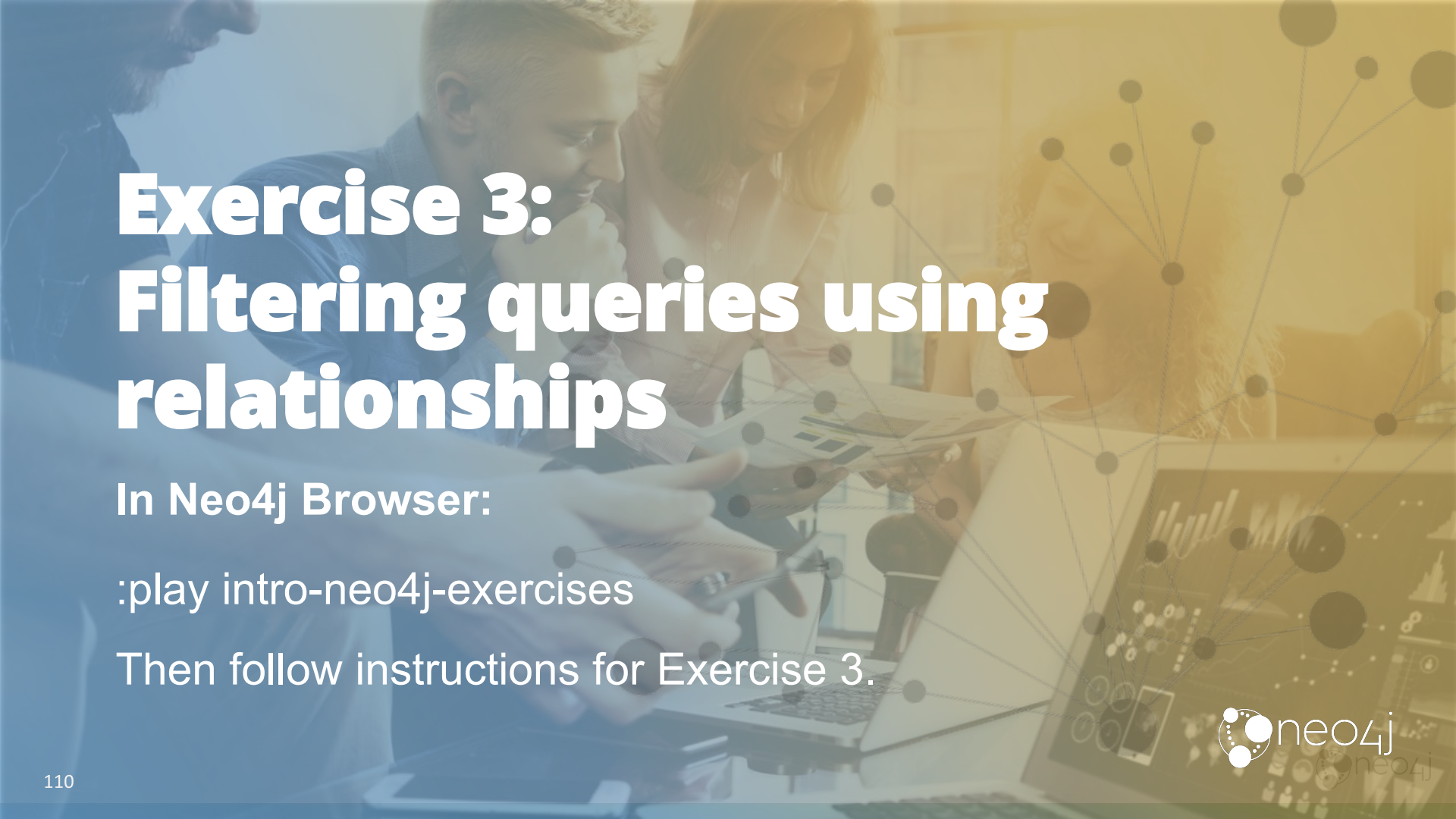
- Node labels are CamelCase and case-sensitive (examples: *Person*, *NetworkAddress*).
- Property keys, variables, parameters, aliases, and functions are camelCase case-sensitive (examples: *businessAddress*, *title*).
- Relationship types are in upper-case and can use the underscore. (examples: *ACTED_IN*,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

Cypher style recommendations - 2

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

- String constants are in single quotes (with exceptions).
- Specify variables only when needed for use later in the Cypher statement.
- Place named nodes and relationships (that use variables) before anonymous nodes and relationships in your MATCH clauses when possible.
- Specify anonymous relationships with -->, --, or <--

```
MATCH (:Person {name: 'Diane Keaton'})-[movRel:ACTED_IN]->
(:Movie {title:"Something's Gotta Give"})
RETURN movRel.roles
```



Exercise 3: Filtering queries using relationships

In Neo4j Browser:

:play intro-neo4j-exercises

Then follow instructions for Exercise 3.


Check your understanding

Question 1

Suppose you have a graph that contains nodes representing customers and other business entities for your application. The node label in the database for a customer is *Customer*. Each *Customer* node has a property named *email* that contains the customer's email address. What Cypher query do you execute to return the email addresses for all customers in the graph?

Select the correct answer.

- ❑ `MATCH (n) RETURN n.Customer.email`
- ❑ `MATCH (c:Customer) RETURN c.email`
- ❑ `MATCH (Customer) RETURN email`
- ❑ `MATCH (c) RETURN Customer.email`

Answer 1

Suppose you have a graph that contains nodes representing customers and other business entities for your application. The node label in the database for a customer is *Customer*. Each *Customer* node has a property named *email* that contains the customer's email address. What Cypher query do you execute to return the email addresses for all customers in the graph?

Select the correct answer.

❑ `MATCH (n) RETURN n.Customer.email`

✓ `MATCH (c:Customer) RETURN c.email`

❑ `MATCH (Customer) RETURN email`

❑ `MATCH (c) RETURN Customer.email`

Question 2

Suppose you have a graph that contains *Customer* and *Product* nodes. A *Customer* node can have a *BOUGHT* relationship with a *Product* node. *Customer* nodes can have other relationships with *Product* nodes. A *Customer* node has a property named *customerName*. A *Product* node has a property named *productName*. What Cypher query do you execute to return all of the products (by name) bought by customer 'ABCCO'.

Select the correct answer.

❑ `MATCH (c:Customer {customerName: 'ABCCO'}) RETURN c.BOUGHT.productName`

❑ `MATCH (:Customer 'ABCCO')-[:BOUGHT]→(p:Product) RETURN p.productName`

❑ `MATCH (p:Product)←[:BOUGHT_BY]-(:Customer 'ABCCO') RETURN p.productName`

❑ `MATCH (:Customer {customerName: 'ABCCO'})-[:BOUGHT]→(p:Product) RETURN p.productName`

Answer 2

Suppose you have a graph that contains *Customer* and *Product* nodes. A *Customer* node can have a *BOUGHT* relationship with a *Product* node. *Customer* nodes can have other relationships with *Product* nodes. A *Customer* node has a property named *customerName*. A *Product* node has a property named *productName*. What Cypher query do you execute to return all of the products (by name) bought by customer 'ABCCO'.

Select the correct answer.

❏ `MATCH (c:Customer {customerName: 'ABCCO'}) RETURN c.BOUGHT.productName`

❏ `MATCH (:Customer 'ABCCO')-[:BOUGHT]→(p:Product) RETURN p.productName`

❏ `MATCH (p:Product)←[:BOUGHT_BY]-(:Customer 'ABCCO') RETURN p.productName`

✓ `MATCH (:Customer {customerName: 'ABCCO'})-[:BOUGHT]→(p:Product) RETURN p.productName`

Question 3

When must you use a variable in a MATCH clause?

Select the correct answer.

- ❏ When you want to query the graph using a node label.
- ❏ When you specify a property value to match the query.
- ❏ When you want to use the node or relationship to return a result.
- ❏ When the query involves two types of nodes.

Answer 3

When must you use a variable in a MATCH clause?

Select the correct answer.

- ❏ When you want to query the graph using a node label.
- ❏ When you specify a property value to match the query.
- ✓ When you want to use the node or relationship to return a result.
- ❏ When the query involves two types of nodes.

Summary

You should be able to write Cypher statements to:

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


Getting More Out of Queries
neo4j
v 1.0

Overview

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

- Filter queries using the `WHERE` clause
- Control query processing
- Control what results are returned
- Work with Cypher lists and dates

Filtering queries using WHERE

Previously you retrieved nodes as follows:

```
MATCH (p:Person)-[:ACTED_IN]->(m:Movie {released: 2008})
RETURN p, m
```

A more flexible syntax for the same query is:

```
MATCH (p:Person)-[:ACTED_IN]->(m:Movie)
WHERE m.released = 2008
RETURN p, m
```

Testing more than equality:

```
MATCH (p:Person)-[:ACTED_IN]->(m:Movie)
WHERE m.released = 2008 OR m.released = 2009
RETURN p, m
```

Specifying ranges in WHERE clauses

This query to find all people who acted in movies released between 2003 and 2004:

```
MATCH (p:Person)-[:ACTED_IN]->(m:Movie)
WHERE m.released >= 2003 AND m.released <= 2004
RETURN p.name, m.title, m.released
```

Is the same as:

```
MATCH (p:Person)-[:ACTED_IN]->(m:Movie)
WHERE 2003 <= m.released <= 2004
RETURN p.name, m.title, m.released
```

Testing labels

These queries:

```
MATCH (p:Person)
RETURN p.name
```

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

Can be rewritten as:

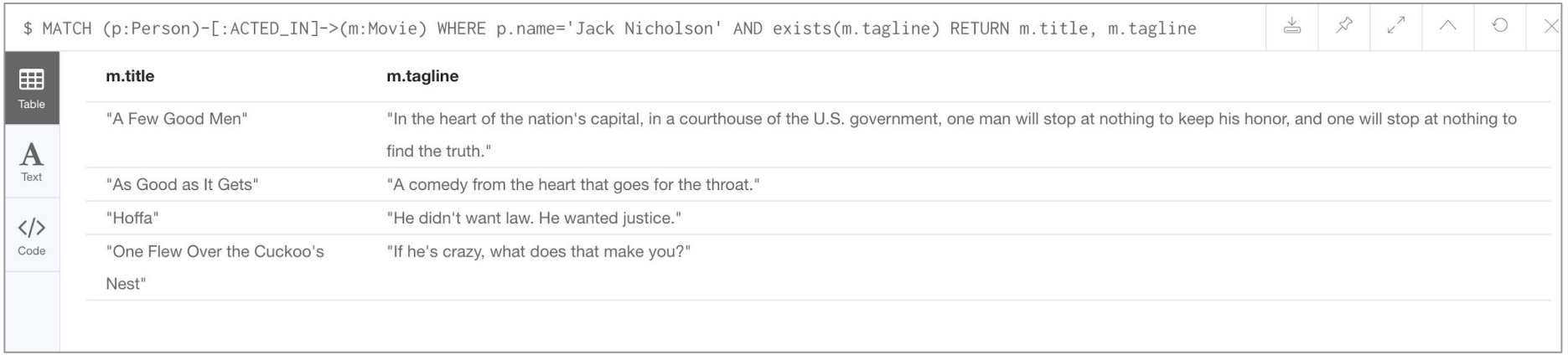
```
MATCH (p)
WHERE p:Person
RETURN p.name
```

```
MATCH (p)-[:ACTED_IN]->(m)
WHERE p:Person AND m:Movie AND m.title='The Matrix'
RETURN p.name
```

Testing the existence of a property

Find all movies that *Jack Nicholson* acted in that have a tagline, returning the title and tagline of the movie:

```
MATCH (p:Person)-[:ACTED_IN]->(m:Movie)
WHERE p.name='Jack Nicholson' AND exists(m.tagline)
RETURN m.title, m.tagline
```

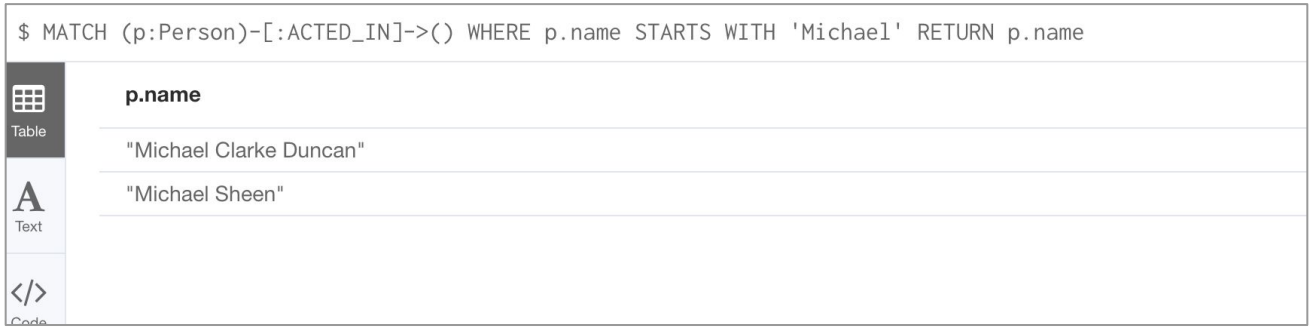


m.title	m.tagline
"A Few Good Men"	"In the heart of the nation's capital, in a courthouse of the U.S. government, one man will stop at nothing to keep his honor, and one will stop at nothing to find the truth."
"As Good as It Gets"	"A comedy from the heart that goes for the throat."
"Hoffa"	"He didn't want law. He wanted justice."
"One Flew Over the Cuckoo's Nest"	"If he's crazy, what does that make you?"

Testing strings

Find all actors whose name begins with *Michael*:

```
MATCH (p:Person)-[:ACTED_IN]->()
WHERE p.name STARTS WITH 'Michael'
RETURN p.name
```

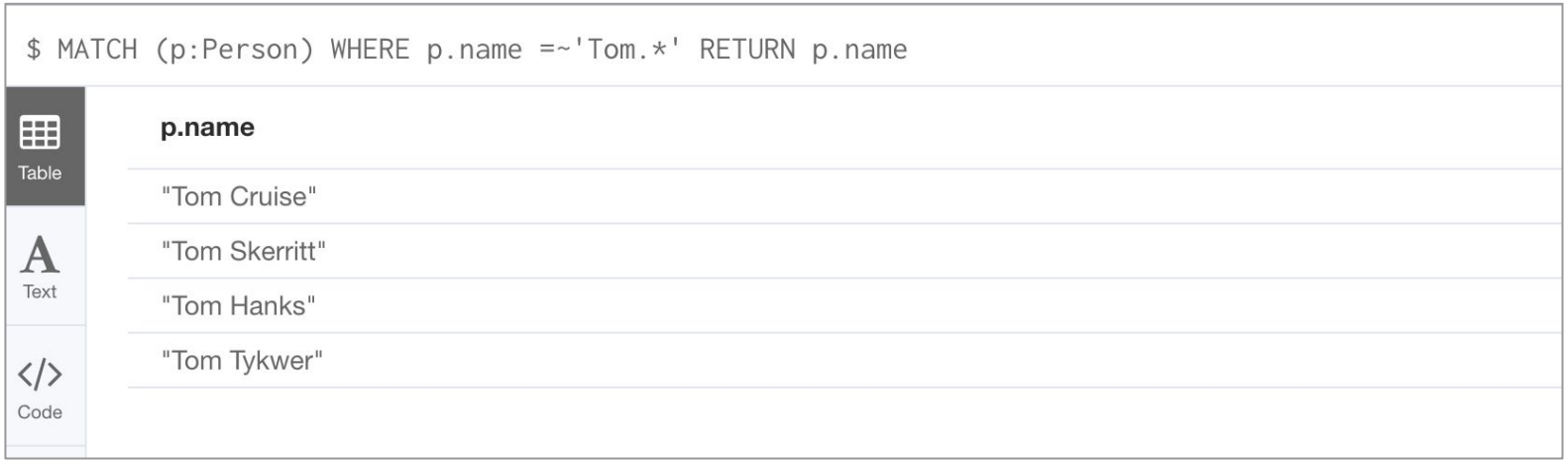



```
MATCH (p:Person)-[:ACTED_IN]->()
WHERE toLower(p.name) STARTS WITH 'michael'
RETURN p.name
```

Testing with regular expressions

Find people whose name starts with *Tom*:

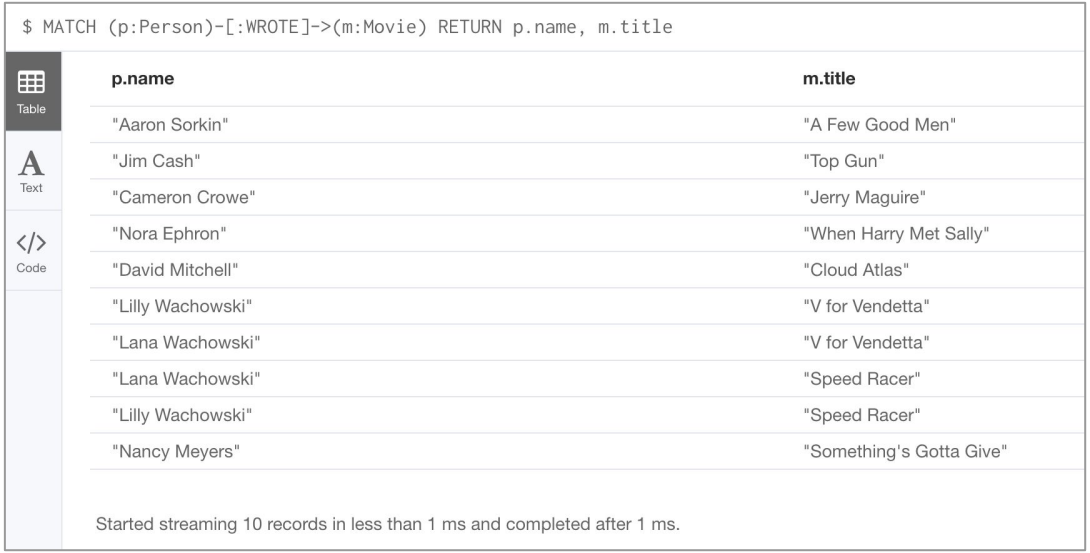
```
MATCH (p:Person)
WHERE p.name =~'Tom.*'
RETURN p.name
```



Testing with patterns - 1

Find all people who wrote movies returning their names and the title of the movie they wrote:

```
MATCH (p:Person)-[:WROTE]->(m:Movie)
RETURN p.name, m.title
```



$ MATCH (p:Person)-[:WROTE]->(m:Movie) RETURN p.name, m.title

Table | Text | Code

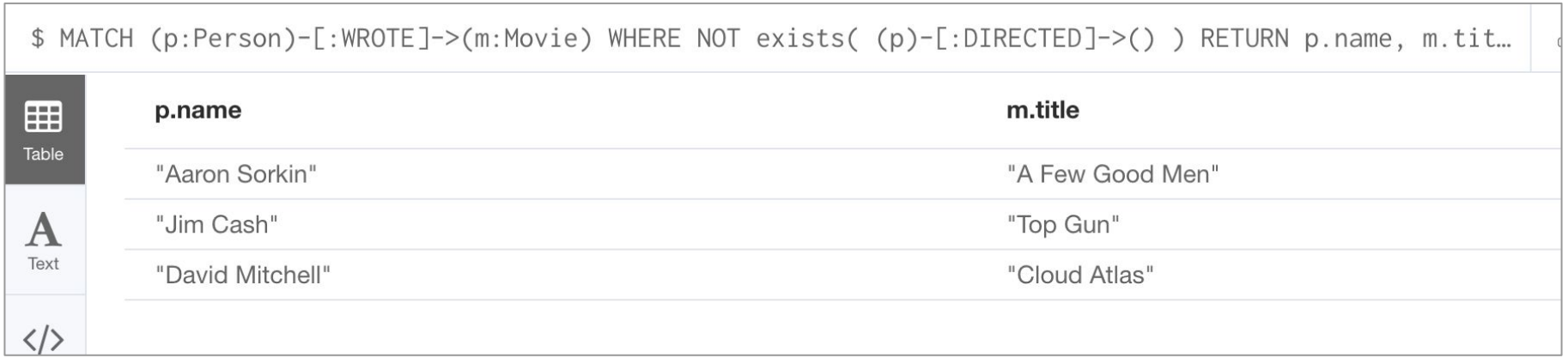
p.name	m.title
"Aaron Sorkin"	"A Few Good Men"
"Jim Cash"	"Top Gun"
"Cameron Crowe"	"Jerry Maguire"
"Nora Ephron"	"When Harry Met Sally"
"David Mitchell"	"Cloud Atlas"
"Lilly Wachowski"	"V for Vendetta"
"Lana Wachowski"	"V for Vendetta"
"Lana Wachowski"	"Speed Racer"
"Lilly Wachowski"	"Speed Racer"
"Nancy Meyers"	"Something's Gotta Give"

Started streaming 10 records in less than 1 ms and completed after 1 ms.

Testing with patterns - 2

Find the people who wrote movies, but did not direct them, returning their names and the title of the movie:

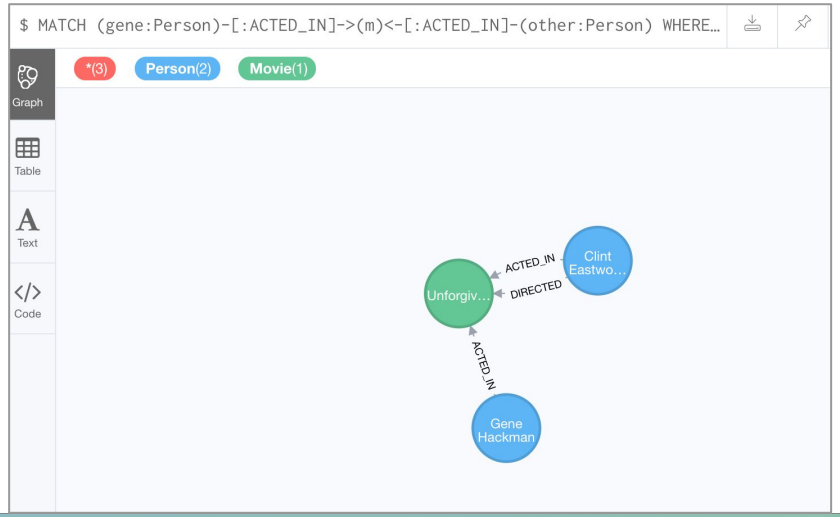
```
MATCH (p:Person)-[:WROTE]->(m:Movie)
WHERE NOT exists( (p)-[:DIRECTED]->() )
RETURN p.name, m.title
```



Testing with patterns - 3

Find *Gene Hackman* and the movies that he acted in with another person who also directed the movie, returning the nodes found:

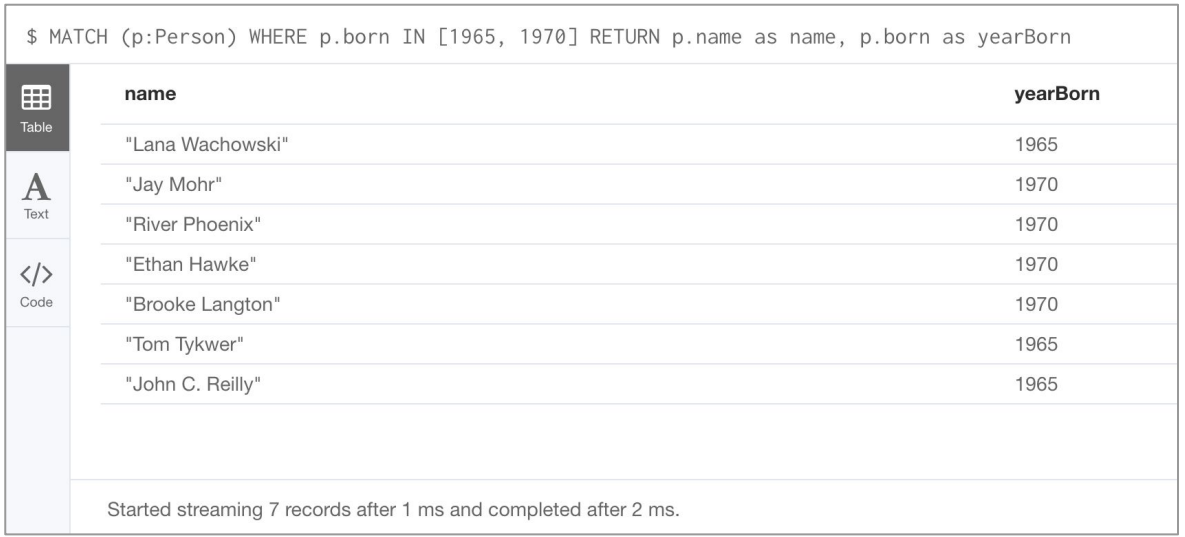
```
MATCH (gene:Person)-[:ACTED_IN]->(m:Movie)<-[:ACTED_IN]-(other:Person)
WHERE gene.name= 'Gene Hackman' AND exists( (other)-[:DIRECTED]->() )
RETURN  gene, other, m
```



Testing with list values - 1

Find all people born in 1965 and 1970:

```
MATCH (p:Person)
WHERE p.born IN [1965, 1970]
RETURN p.name as name, p.born as yearBorn
```



Testing with list values - 2

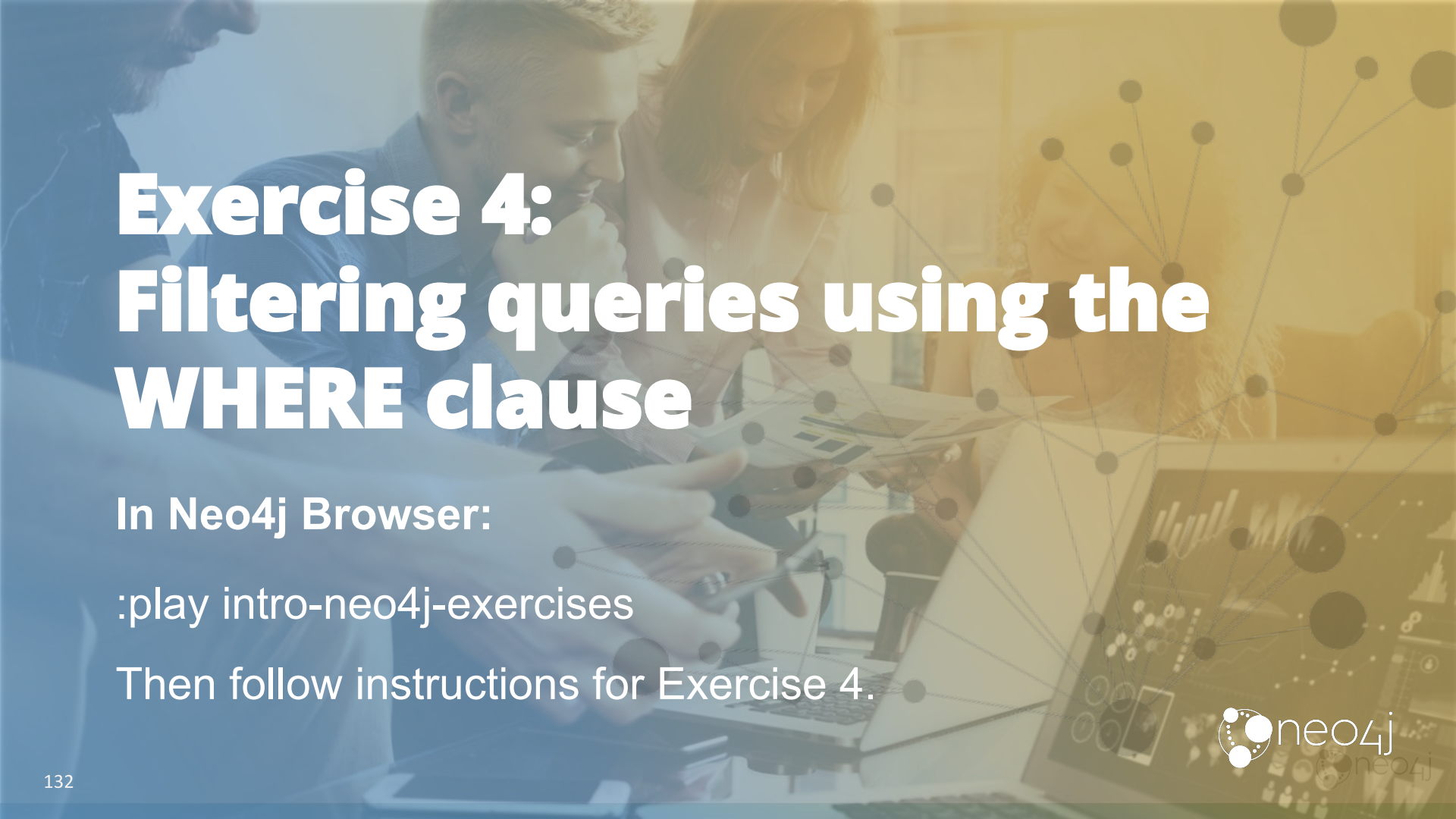
Find the actor who played *Neo* in the movie, *The Matrix*:

```
MATCH (p:Person)-[r:ACTED_IN]->(m:Movie)
WHERE  'Neo' IN r.roles AND m.title='The Matrix'
RETURN p.name
```

```
$ MATCH (p:Person)-[r:ACTED_IN]->(m:Movie) WHERE "Neo" IN r.roles and m.title="The Matrix" RETURN p.name
```

Table

p.name
"Keanu Reeves"



Exercise 4: Filtering queries using the WHERE clause

In Neo4j Browser:

:play intro-neo4j-exercises

Then follow instructions for Exercise 4.

Controlling query processing

- Multiple MATCH clauses
- Path variables
- Varying length paths
- Finding the shortest path
- Optional pattern matching
- Collecting results into lists
- Counting results
- Using the WITH clause to control processing

Specifying multiple MATCH patterns

This query to find people who either acted or directed a movie released in *2000* is specified with two MATCH patterns:

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie),
      (m:Movie)<-[:DIRECTED]-(d:Person)
WHERE m.released = 2000
RETURN a.name, m.title, d.name
```

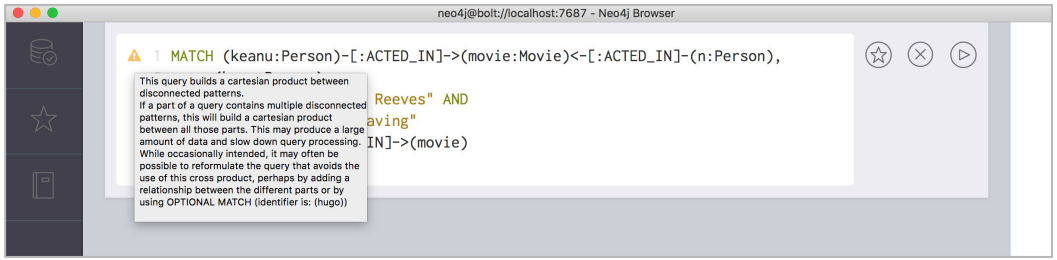
A best practice is to use a single MATCH pattern if possible:

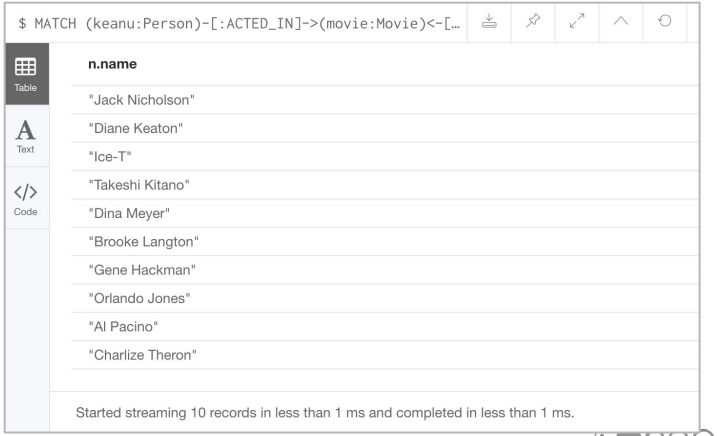
```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)<-[:DIRECTED]-(d:Person)
WHERE m.released = 2000
RETURN a.name, m.title, d.name
```

Example 1: Using two MATCH patterns

Find the actors who acted in the same movies as *Keanu Reeves*, but not when *Hugo Weaving* acted in the same movie:

```
MATCH (keanu:Person)-[:ACTED_IN]->(movie:Movie)<-[:ACTED_IN]-(n:Person), (hugo:Person)
WHERE keanu.name='Keanu Reeves' AND hugo.name='Hugo Weaving' AND
      NOT (hugo)-[:ACTED_IN]->(movie)
RETURN n.name
```





Example 2: Using two MATCH patterns

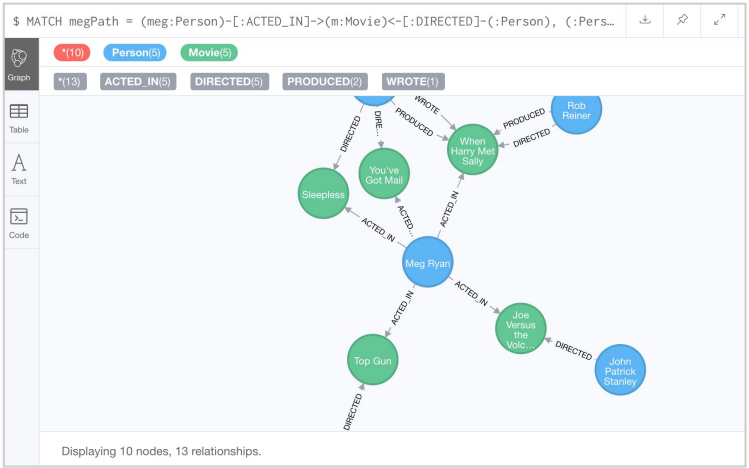
Retrieve the movies that *Meg Ryan* acted in and their respective directors, as well as the other actors that acted in these movies:

```
MATCH (meg:Person)-[:ACTED_IN]->(m:Movie)<-[:DIRECTED]-(d:Person),
      (other:Person)-[:ACTED_IN]->(m)
WHERE meg.name = 'Meg Ryan'
RETURN m.title as movie, d.name AS director , other.name AS `co-actors`
```

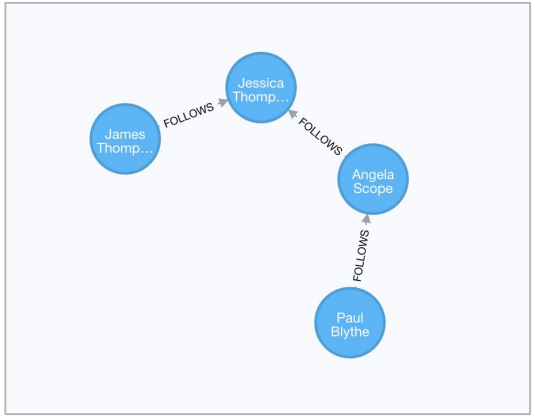
Setting path variables

Path variables allow you to reuse path/pattern in a query or return the path. Here us the previous query where the path is returned.

```
MATCH megPath =
(meg:Person)-[:ACTED_IN]->(m:Movie)<-[:DIRECTED]-(:Person),
      (:Person)-[:ACTED_IN]->(m)
WHERE meg.name = 'Meg Ryan'
RETURN megPath
```



Specifying varying length paths



Find all people who are exactly two hops away from *Paul Blythe*:

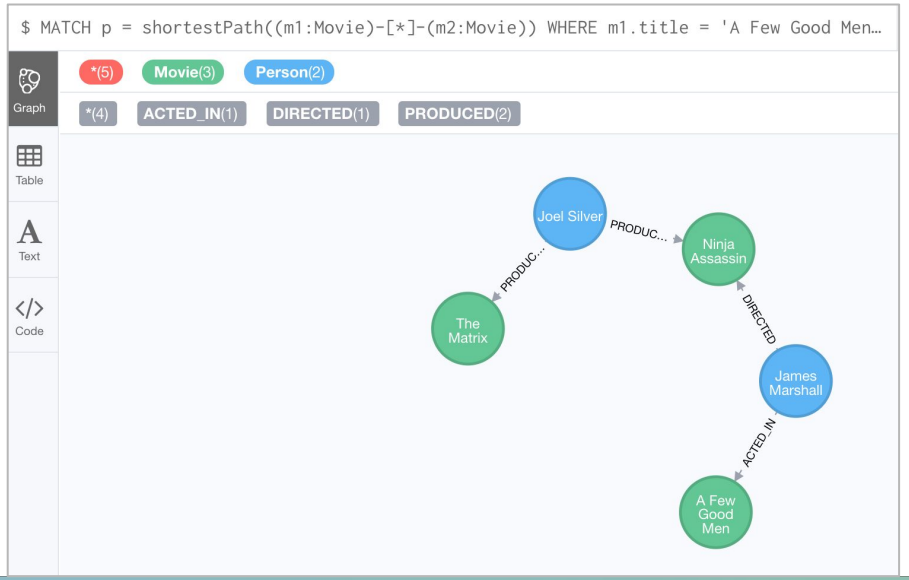
```
MATCH (follower:Person)-[:FOLLOWS*2]->(p:Person)
WHERE follower.name = 'Paul Blythe'
RETURN p
```

Finding the shortest path

Find the shortest path between the movies *The Matrix* and *A Few Good Men*:

```
MATCH p = shortestPath((m1:Movie)-[*]-(m2:Movie))
WHERE m1.title = 'A Few Good Men' AND
      m2.title = 'The Matrix'
RETURN  p
```

Specifying * for the relationship means we use any relationship type for determining the path.

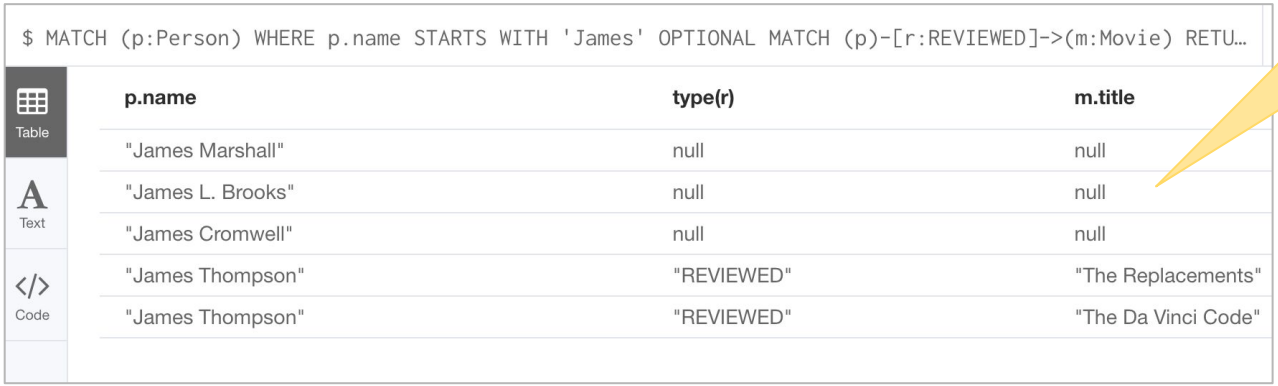


neo4j

Specifying optional pattern matching

Find all people whose name starts with *James*, additionally return people who have reviewed a movie:

```
MATCH (p:Person)
WHERE p.name STARTS WITH 'James'
OPTIONAL MATCH (p)-[r:REVIEWED]->(m:Movie)
RETURN p.name, type(r), m.title
```

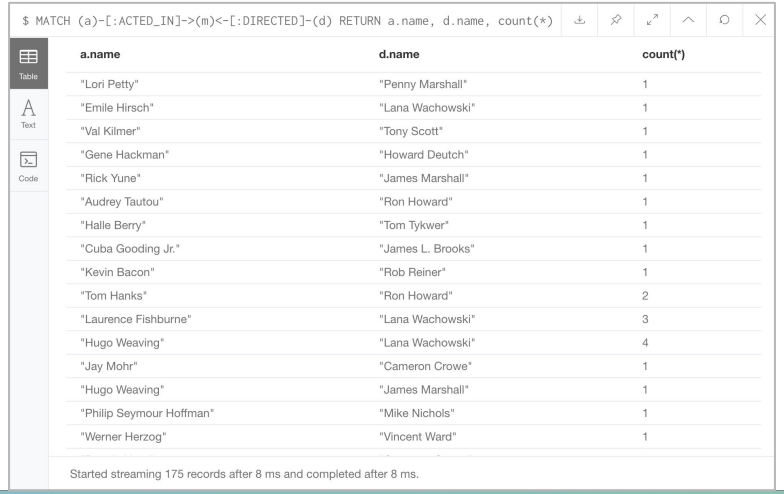


Nulls are returned for the missing parts of the pattern. Similar to an outer join in SQL.

Aggregation in Cypher

- Different from SQL - no need to specify a grouping key.
- As soon as you use an aggregation function, all non-aggregated result columns automatically become grouping keys.
- Implicit grouping based upon fields in the RETURN clause.

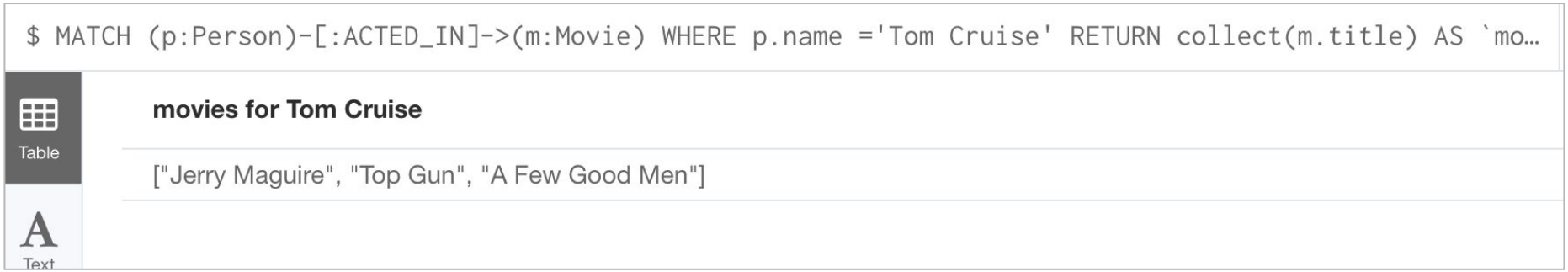
```
// implicitly groups by a.name and d.name
MATCH (a)-[:ACTED_IN]->(m)<-[:DIRECTED]-(d)
RETURN a.name, d.name, count(*)
```



Collecting results

Find the movies that Tom Cruise acted in and return them as a list:

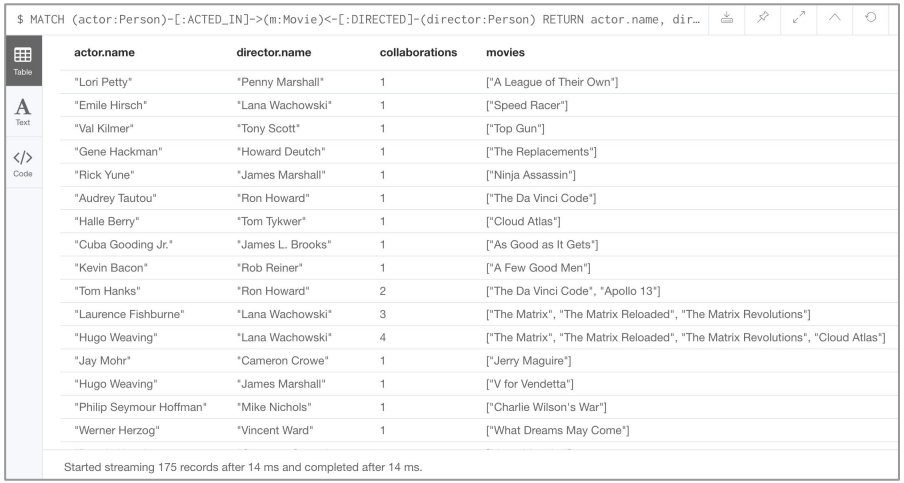
```
MATCH (p:Person)-[:ACTED_IN]->(m:Movie)
WHERE p.name ='Tom Cruise'
RETURN collect(m.title) AS `movies for Tom Cruise`
```



Counting results

Find all of the actors and directors who worked on a movie, return the count of the number paths found between actors and directors and collect the movies as a list:

```
MATCH (actor:Person)-[:ACTED_IN]->(m:Movie)<-[:DIRECTED]-(director:Person)
RETURN actor.name, director.name,  count(m) AS collaborations,
       collect(m.title) AS movies
```

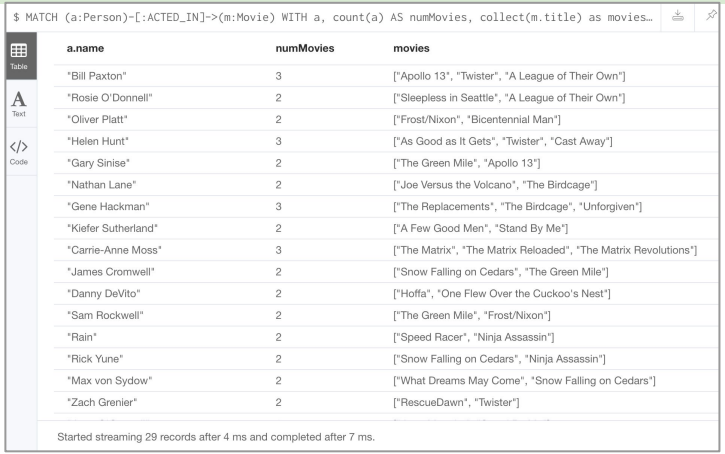


Additional processing using WITH - 1

Use the WITH clause to perform intermediate processing or data flow operations.

Find all actors who acted in two or three movies, return the list of movies:

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
WITH  a, count(a) AS numMovies, collect(m.title) as movies
WHERE numMovies > 1 AND numMovies < 4
RETURN a.name, numMovies, movies
```



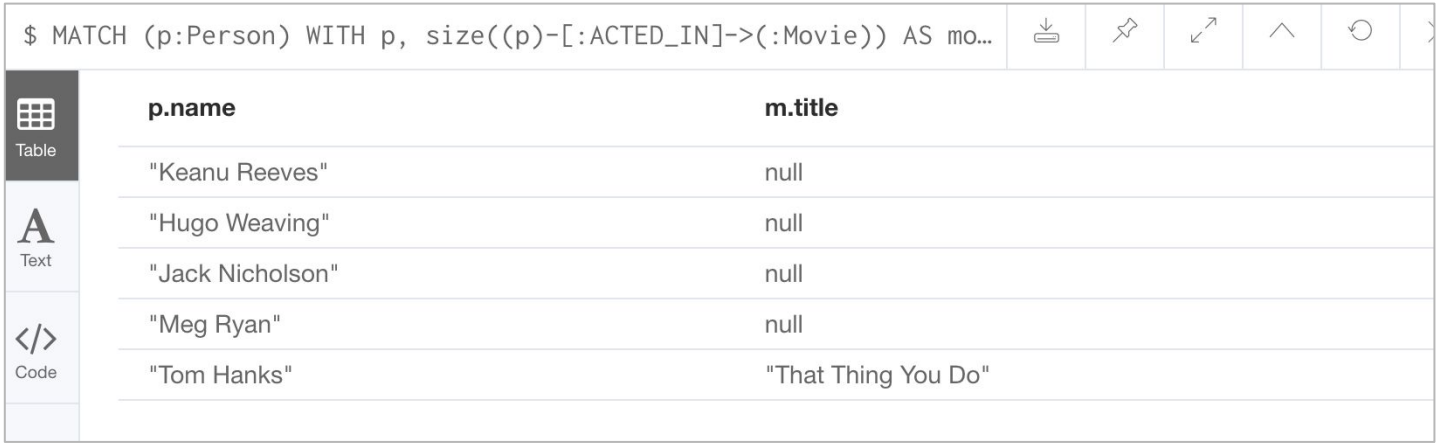
a.name	numMovies	movies
"Bill Paxton"	3	["Apollo 13", "Twister", "A League of Their Own"]
"Rosie O'Donnell"	2	["Sleepless in Seattle", "A League of Their Own"]
"Oliver Platt"	2	["Frost/Nixon", "Bicentennial Man"]
"Helen Hunt"	3	["As Good as It Gets", "Twister", "Cast Away"]
"Gary Sinise"	2	["The Green Mile", "Apollo 13"]
"Nathan Lane"	2	["Joe Versus the Volcano", "The Birdcage"]
"Gene Hackman"	3	["The Replacements", "The Birdcage", "Unforgiven"]
"Kiefer Sutherland"	2	["A Few Good Men", "Stand By Me"]
"Carrie-Anne Moss"	3	["The Matrix", "The Matrix Reloaded", "The Matrix Revolutions"]
"James Cromwell"	2	["Snow Falling on Cedars", "The Green Mile"]
"Danny DeVito"	2	["Hoffa", "One Flew Over the Cuckoo's Nest"]
"Sam Rockwell"	2	["The Green Mile", "Frost/Nixon"]
"Rain"	2	["Speed Racer", "Ninja Assassin"]
"Rick Yune"	2	["Snow Falling on Cedars", "Ninja Assassin"]
"Max von Sydow"	2	["What Dreams May Come", "Snow Falling on Cedars"]
"Zach Grenier"	2	["RescueDawn", "Twister"]

Started streaming 29 records after 4 ms and completed after 7 ms.

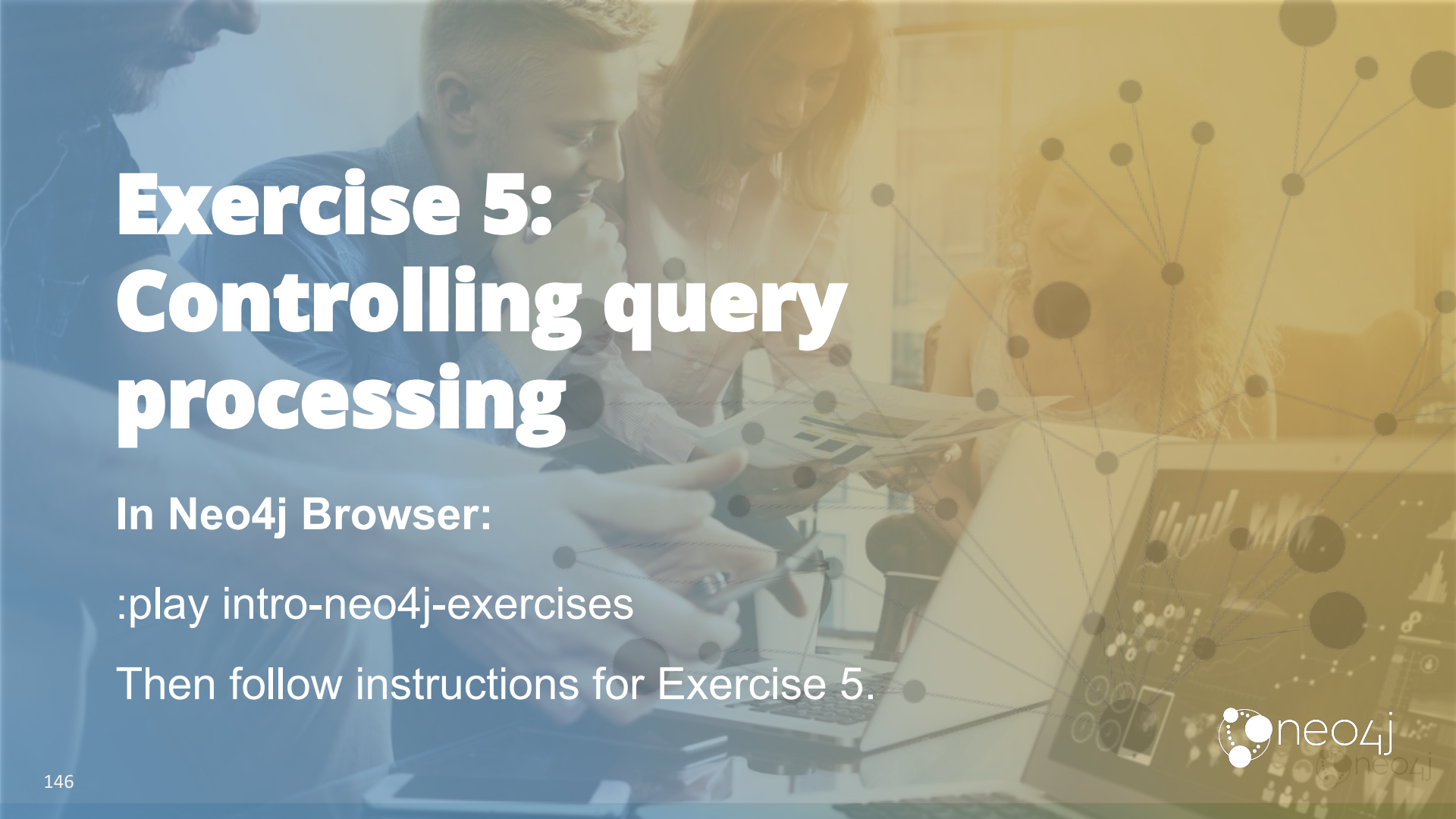
Additional processing using WITH - 2

Find all actors who have acted in at least five movies, and find (optionally) the movies they directed and return the person and those movies.:

```
MATCH (p:Person)
WITH p, size((p)-[:ACTED_IN]->(:Movie)) AS movies
WHERE movies >= 5
OPTIONAL MATCH (p)-[:DIRECTED]->(m:Movie)
RETURN p.name, m.title
```



p.name	m.title
"Keanu Reeves"	null
"Hugo Weaving"	null
"Jack Nicholson"	null
"Meg Ryan"	null
"Tom Hanks"	"That Thing You Do"



Exercise 5: Controlling query processing

In Neo4j Browser:

:play intro-neo4j-exercises

Then follow instructions for Exercise 5.

Controlling how results are returned

- Eliminating duplication
- Ordering results
- Limiting the number of results

Eliminating duplication - 1

Here is a query where the movie *That Thing You Do* is repeated in the list because Tom Hanks both acted in and directed the movie:

```
MATCH (p:Person)-[:DIRECTED | :ACTED_IN]->(m:Movie)
WHERE p.name = 'Tom Hanks'
RETURN m.released, collect(m.title) AS movies
```

Eliminating duplication - 2

We can eliminate the duplication in this query by specifying the DISTINCT keyword as follows:

```
MATCH (p:Person)-[:DIRECTED | :ACTED_IN]->(m:Movie)
WHERE p.name = 'Tom Hanks'
RETURN m.released, collect( DISTINCT m.title) AS movies
```

Using WITH and DISTINCT to eliminate duplication

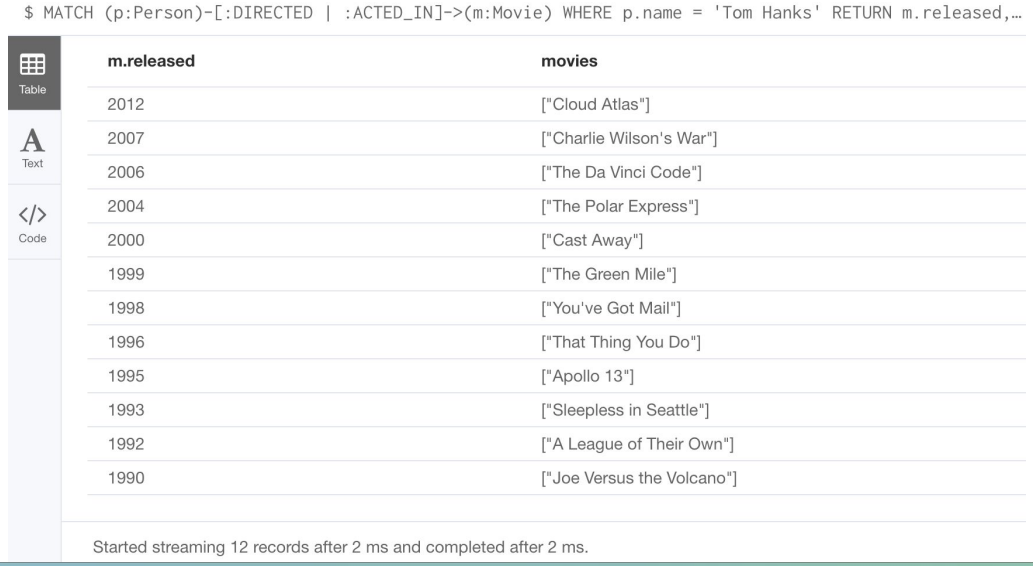
We can also eliminate the duplication in this query by specifying WITH DISTINCT as follows:

```
MATCH (p:Person)-[:DIRECTED | :ACTED_IN]->(m:Movie)
WHERE p.name = 'Tom Hanks'
WITH DISTINCT m
RETURN m.released, m.title
```

Ordering results

You can return results in order based upon the property value:

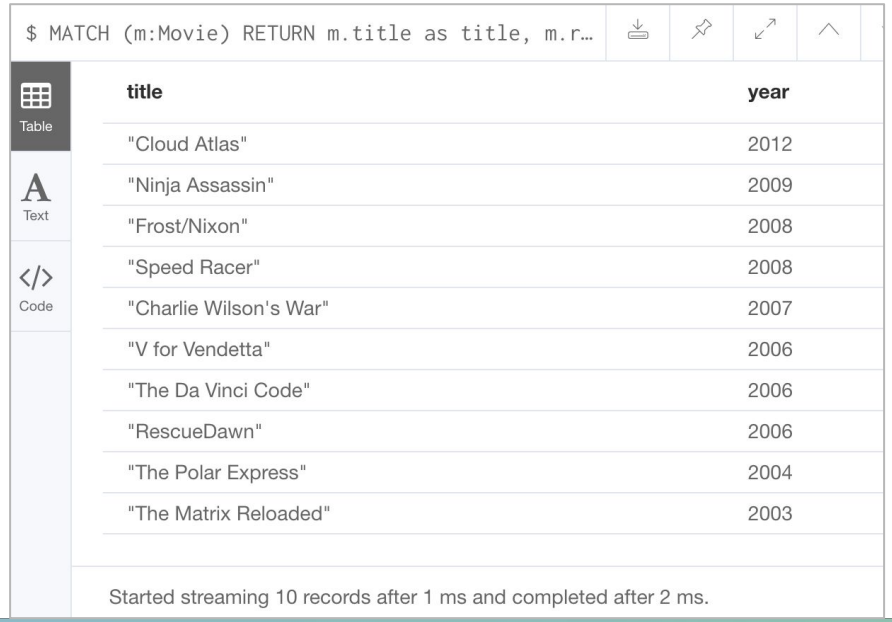
```
MATCH (p:Person)-[:DIRECTED | :ACTED_IN]->(m:Movie)
WHERE p.name = 'Tom Hanks'
RETURN m.released, collect(DISTINCT m.title) AS movies
       ORDER BY m.released DESC
```



Limiting results

What are the titles of the ten most recently released movies:

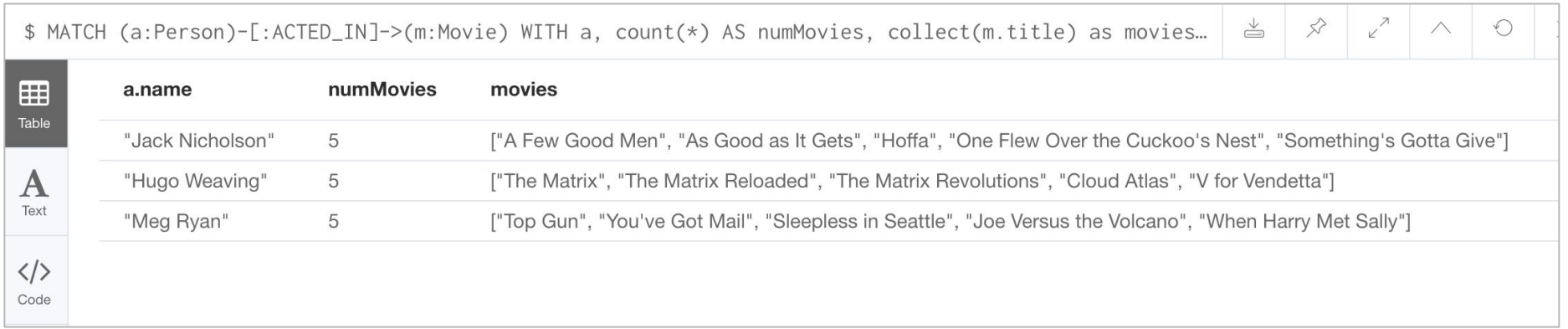
```
MATCH (m:Movie)
RETURN m.title as title, m.released as year ORDER BY m.released
DESC LIMIT 10
```



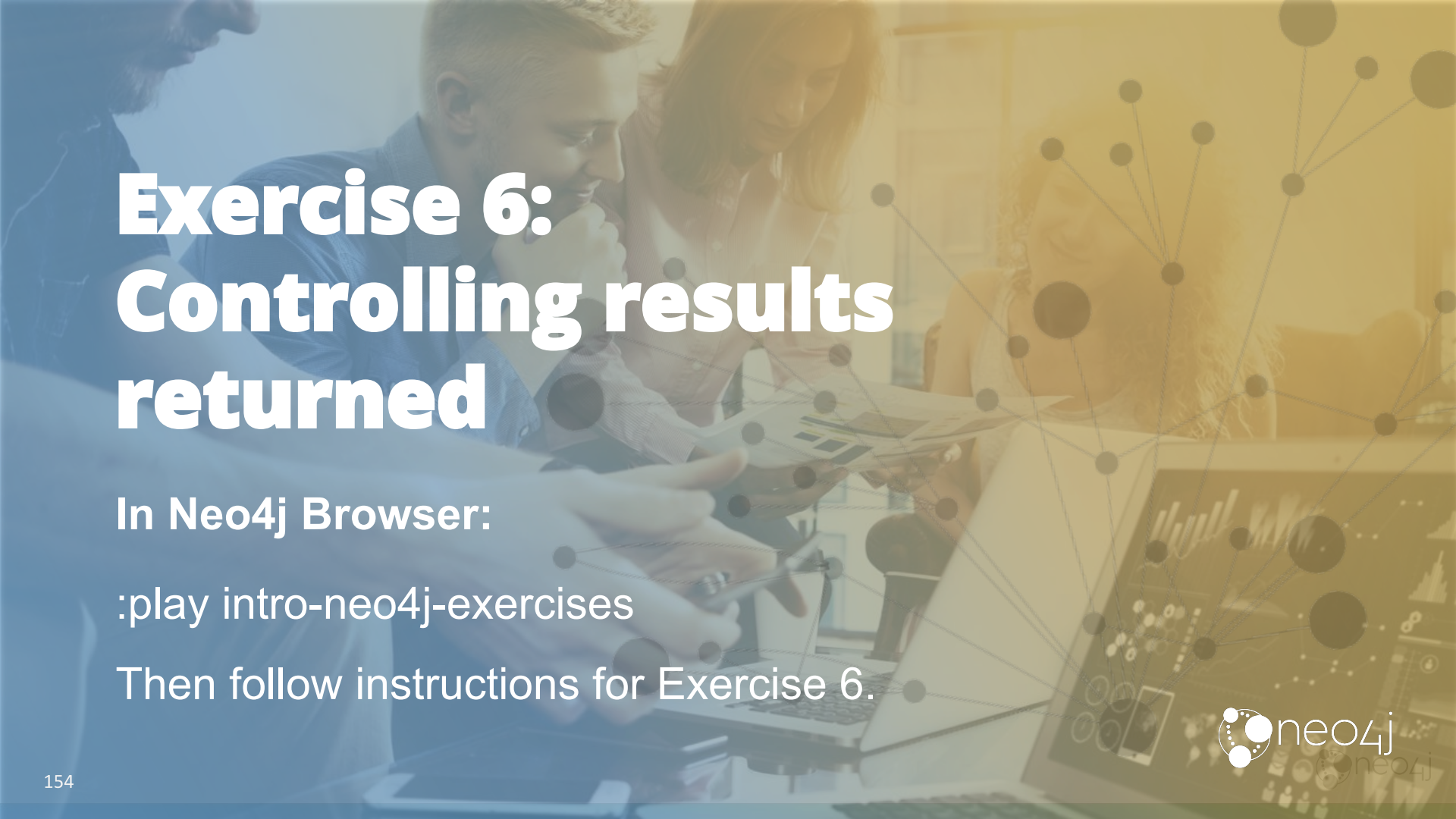
Controlling number of results using WITH

Retrieve the actors who have acted in exactly five movies, returning the list of movies:

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
WITH a, count(*) AS numMovies, collect(m.title) as movies
WHERE numMovies = 5
RETURN a.name, numMovies, movies
```



a.name	numMovies	movies
"Jack Nicholson"	5	["A Few Good Men", "As Good as It Gets", "Hoffa", "One Flew Over the Cuckoo's Nest", "Something's Gotta Give"]
"Hugo Weaving"	5	["The Matrix", "The Matrix Reloaded", "The Matrix Revolutions", "Cloud Atlas", "V for Vendetta"]
"Meg Ryan"	5	["Top Gun", "You've Got Mail", "Sleepless in Seattle", "Joe Versus the Volcano", "When Harry Met Sally"]



Exercise 6: Controlling results returned

In Neo4j Browser:

:play intro-neo4j-exercises

Then follow instructions for Exercise 6.

Working with Cypher data

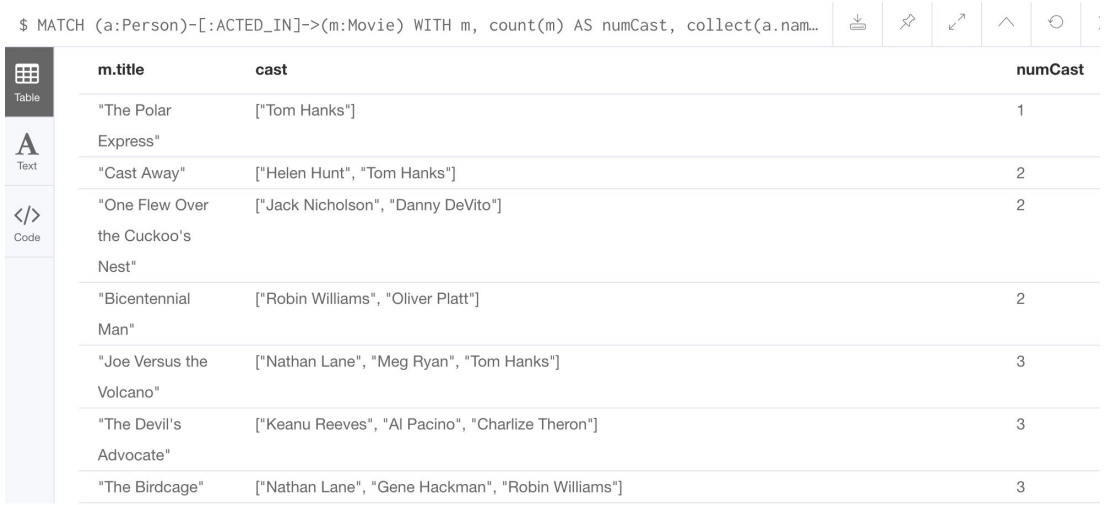
Properties do not have types, but the values of the properties can contain data of any types:

- String ‘Tom Cruise’
- Numeric 2012
- Date 2018-12-15
- Spatial {x: 12.0, y: 56.0, z: 1000.0}
- List [‘Mon’, ’Tue’, ’Wed’, ’Thu’, ‘Fri’, ‘Sat’, ‘Sun’]
- Map [Q1: 395, Q2: 200, Q3: 604, Q3: 509]

Lists

Retrieve all actors and their movies, returning each movie, its cast, and the size of the cast, ordered by the size of the cast:

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
WITH  m, count(m) AS numCast,  collect(a.name) as cast
RETURN m.title, cast, numCast ORDER BY size(cast)
```



m.title	cast	numCast
"The Polar Express"	["Tom Hanks"]	1
"Cast Away"	["Helen Hunt", "Tom Hanks"]	2
"One Flew Over the Cuckoo's Nest"	["Jack Nicholson", "Danny DeVito"]	2
"Bicentennial Man"	["Robin Williams", "Oliver Platt"]	2
"Joe Versus the Volcano"	["Nathan Lane", "Meg Ryan", "Tom Hanks"]	3
"The Devil's Advocate"	["Keanu Reeves", "Al Pacino", "Charlize Theron"]	3
"The Birdcage"	["Nathan Lane", "Gene Hackman", "Robin Williams"]	3

Unwinding lists

Just as you can create lists, you can take them apart as separate data values:

```
WITH [1, 2, 3] AS list
UNWIND list AS row
RETURN list, row
```

```
$ WITH [1, 2, 3] AS list UNWIND list AS row RETURN list, row
```

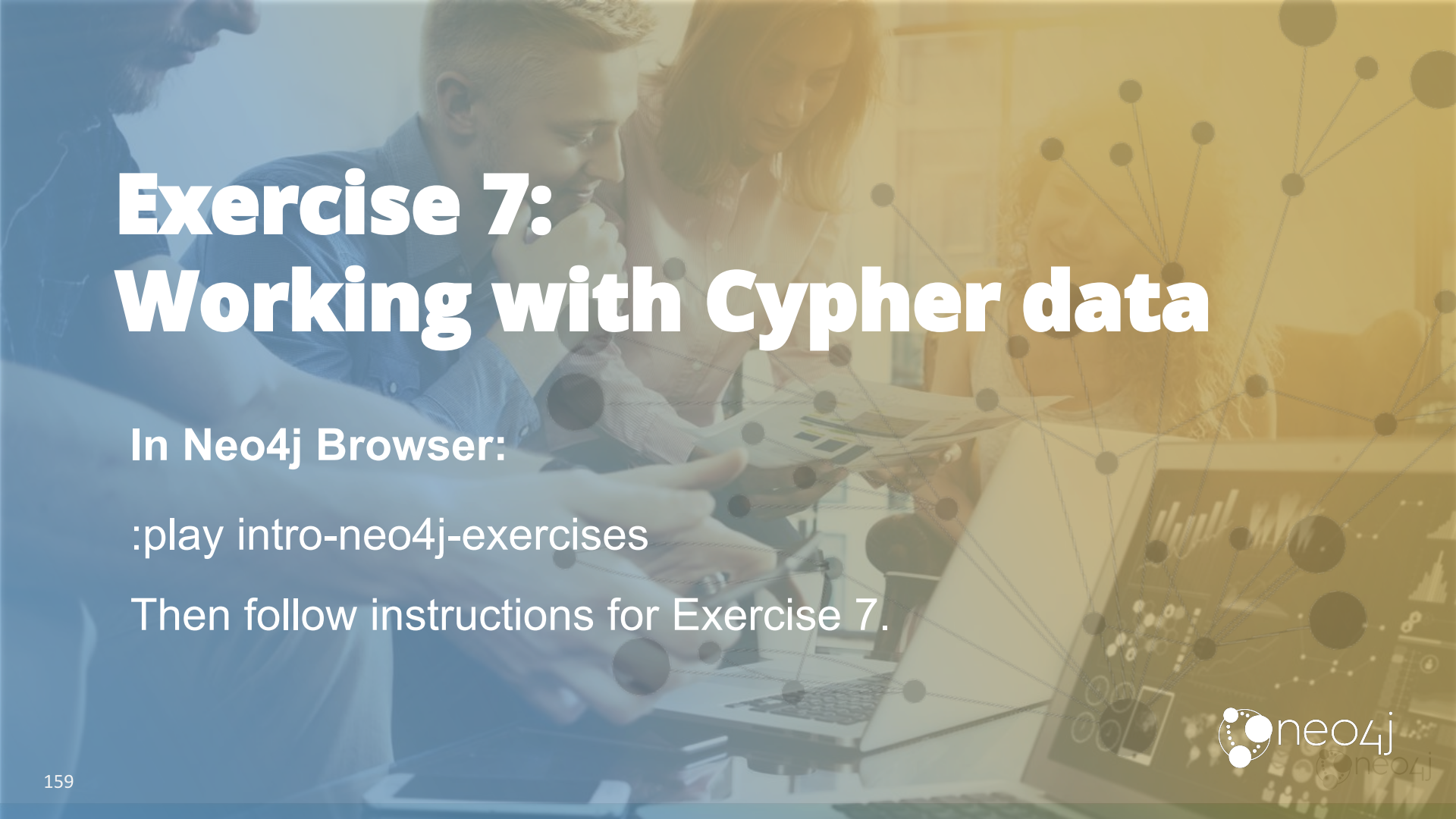
Table | Text | </>

list	row
[1, 2, 3]	1
[1, 2, 3]	2
[1, 2, 3]	3

Dates

Find all actors in the graph that have a value for *born* and calculate their ages as of today:

```
MATCH (actor:Person)-[:ACTED_IN]->(:Movie)
WHERE exists(actor.born)
// calculate the age
with DISTINCT actor, date().year  - actor.born as age
RETURN actor.name, age as `age today`
      ORDER BY actor.born DESC
```



Exercise 7: Working with Cypher data

In Neo4j Browser:

:play intro-neo4j-exercises

Then follow instructions for Exercise 7.


Check your understanding

Question 1

Suppose you want to add a `WHERE` clause at the end of this statement to filter the results retrieved.

```
MATCH (p:Person)-[rel]->(m:Movie)<-[:PRODUCED]-(:Person)
```

What variables, can you test in the `WHERE` clause:

Select the correct answers.

- ❑ p
- ❑ rel
- ❑ m
- ❑ PRODUCED

Answer 1

Suppose you want to add a `WHERE` clause at the end of this statement to filter the results retrieved.

```
MATCH (p:Person)-[rel]->(m:Movie)<-[:PRODUCED]-(:Person)
```

What variables, can you test in the `WHERE` clause:

Select the correct answers.

- ✓ p
- ✓ rel
- ✓ m
- ❑ PRODUCED

Question 2

Suppose you want to retrieve all movies that have a *released* property value that is 2000, 2002, 2004, 2006, or 2008. Here is an incomplete Cypher example to return the *title* property values of all movies released in these years.

```
MATCH (m:MovMATCH (m:Movie)
WHERE m.released XX [2000, 2002, 2004, 2006, 2008]
RETURN m.title
```

What keyword do you specify for **XX**?

Select the correct answer.

- ❑ CONTAINS
- ❑ IN
- ❑ IS
- ❑ EQUALS

Answer 2

Suppose you want to retrieve all movies that have a *released* property value that is 2000, 2002, 2004, 2006, or 2008. Here is an incomplete Cypher example to return the *title* property values of all movies released in these years.

```
MATCH (m:MovMATCH (m:Movie)
WHERE m.released XX [2000, 2002, 2004, 2006, 2008]
RETURN m.title
```

What keyword do you specify for **XX**?

Select the correct answer.

- ❏ CONTAINS
- ✓ IN
- ❏ IS
- ❏ EQUALS

Question 3

Given this Cypher query:

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
WITH  m, count(m) AS numMovies, collect(m.title) as movies
WHERE numMovies > 1 AND numMovies < 4
RETURN //??
```

What variables or aliases can be used to return values?

Select the correct answers.

- ❑ a
- ❑ m
- ❑ numMovies
- ❑ movies

Answer 3

Given this Cypher query:

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
WITH  m, count(m) AS numMovies, collect(m.title) as movies
WHERE numMovies > 1 AND numMovies < 4
RETURN //??
```

What variables or aliases can be used to return values?

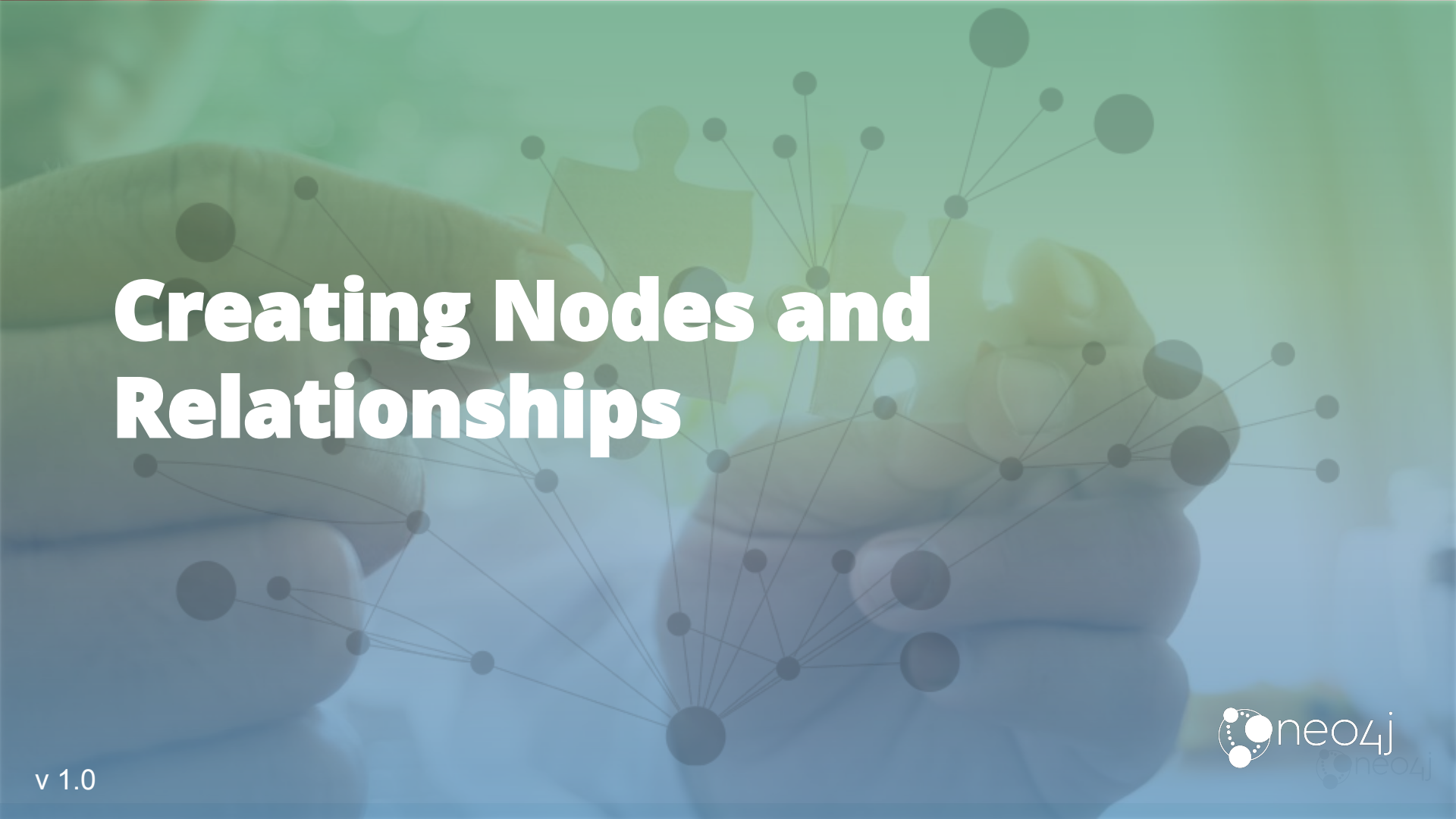
Select the correct answers.

- ❏ a
- ✓ m
- ✓ numMovies
- ✓ movies

Summary

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

- Filter queries using the `WHERE` clause
- Control query processing
- Control what results are returned
- Work with Cypher lists and dates


Creating Nodes and Relationships
neo4j
v 1.0

Overview

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

Create a node of type *Movie* with the *title* property set to *Batman Begins*:

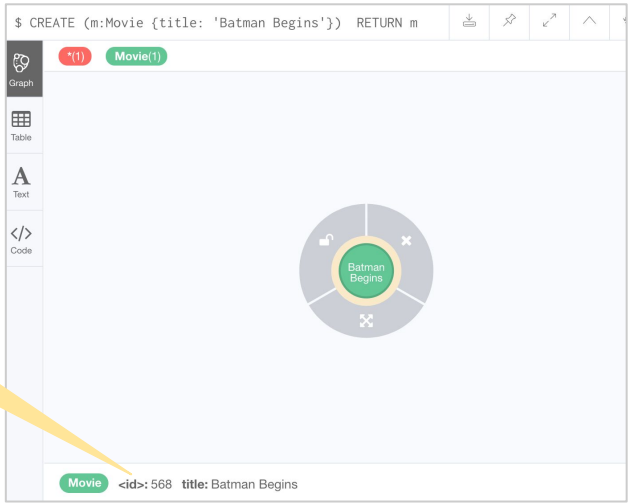
```
CREATE (:Movie {title: 'Batman Begins'})
```

Create a node of type *Movie* and *Action* with the *title* property set to *Batman Begins*:

```
CREATE (:Movie:Action {title: 'Batman Begins'})
```

Create a node of type *Movie* with the *title* property set to *Batman Begins* and return the node:

```
CREATE (m:Movie {title: 'Batman Begins'})
RETURN m
```

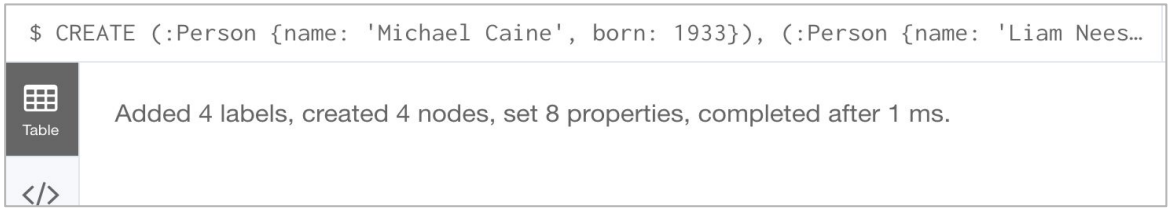


<id> is set by the graph engine

Creating multiple nodes

Create some *Person* nodes for actors and the director for the movie, *Batman Begins*:

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



Important: The graph engine 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:

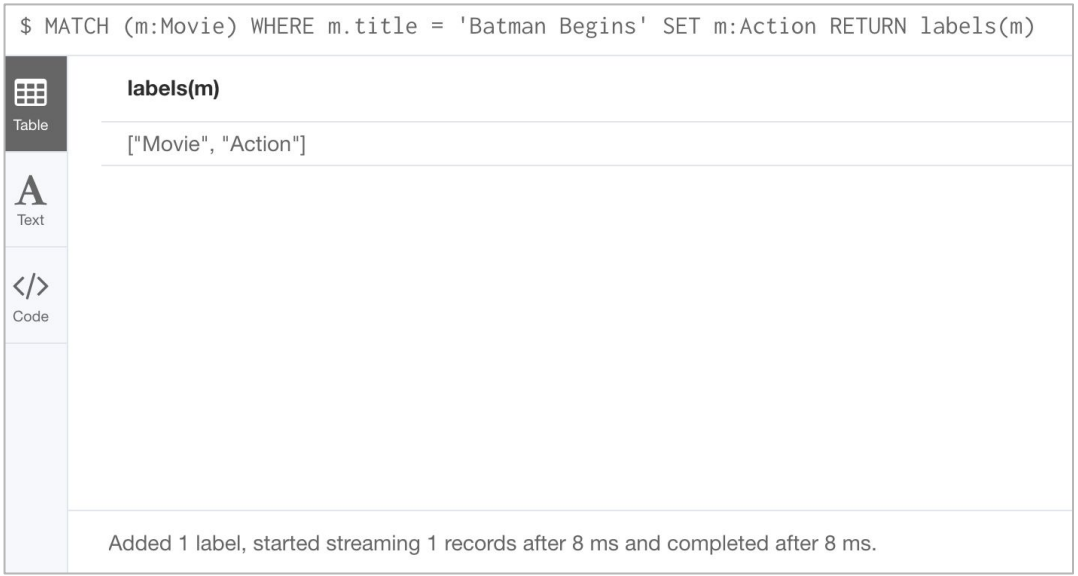
1. You can use `MERGE` rather than `CREATE` when creating the node.

2. You can add constraints to your graph.

Adding a label to a node

Add the *Action* label to the movie, *Batman Begins*, return all labels for this node:

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



Removing a label from a node

Remove the *Action* label to the movie, *Batman Begins*, return all labels for this node:

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

Adding or updating properties for a node

- If property does not exist for the node, it is added with the specified value.
- If property exists for the node, it is updated with the specified value

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
```

Adding properties to a node - JSON style

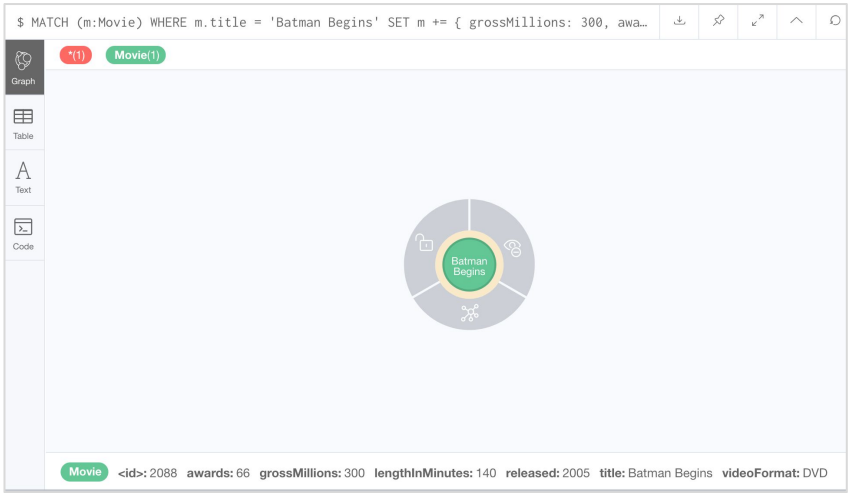
Add or update all properties: *title*, *released*, *lengthInMinutes*, *videoFormat*, and *grossMillions* for the movie *Batman Begins*:

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

Adding or updating properties for a node - JSON style

Add the *awards* property and update the *grossMillions* for the movie *Batman Begins*:

```
MATCH (m:Movie)
WHERE m.title = 'Batman Begins'
SET  m += { grossMillions: 300,
            awards: 66}
RETURN m
```



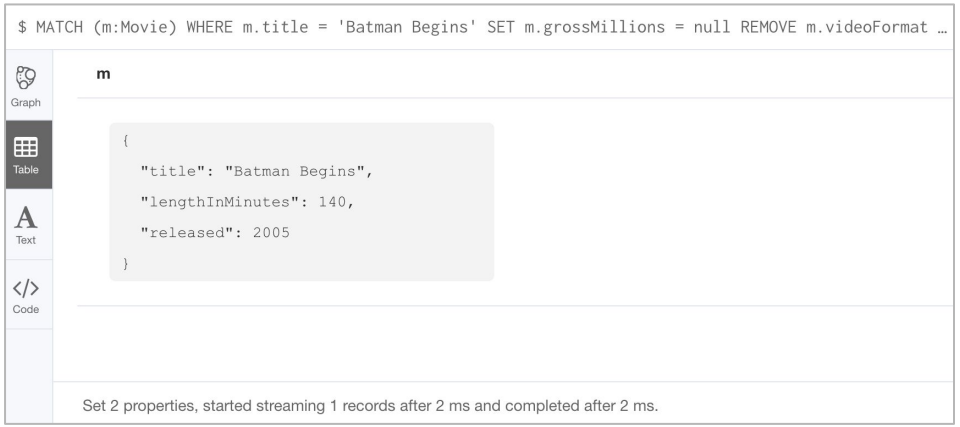
Removing properties from a node

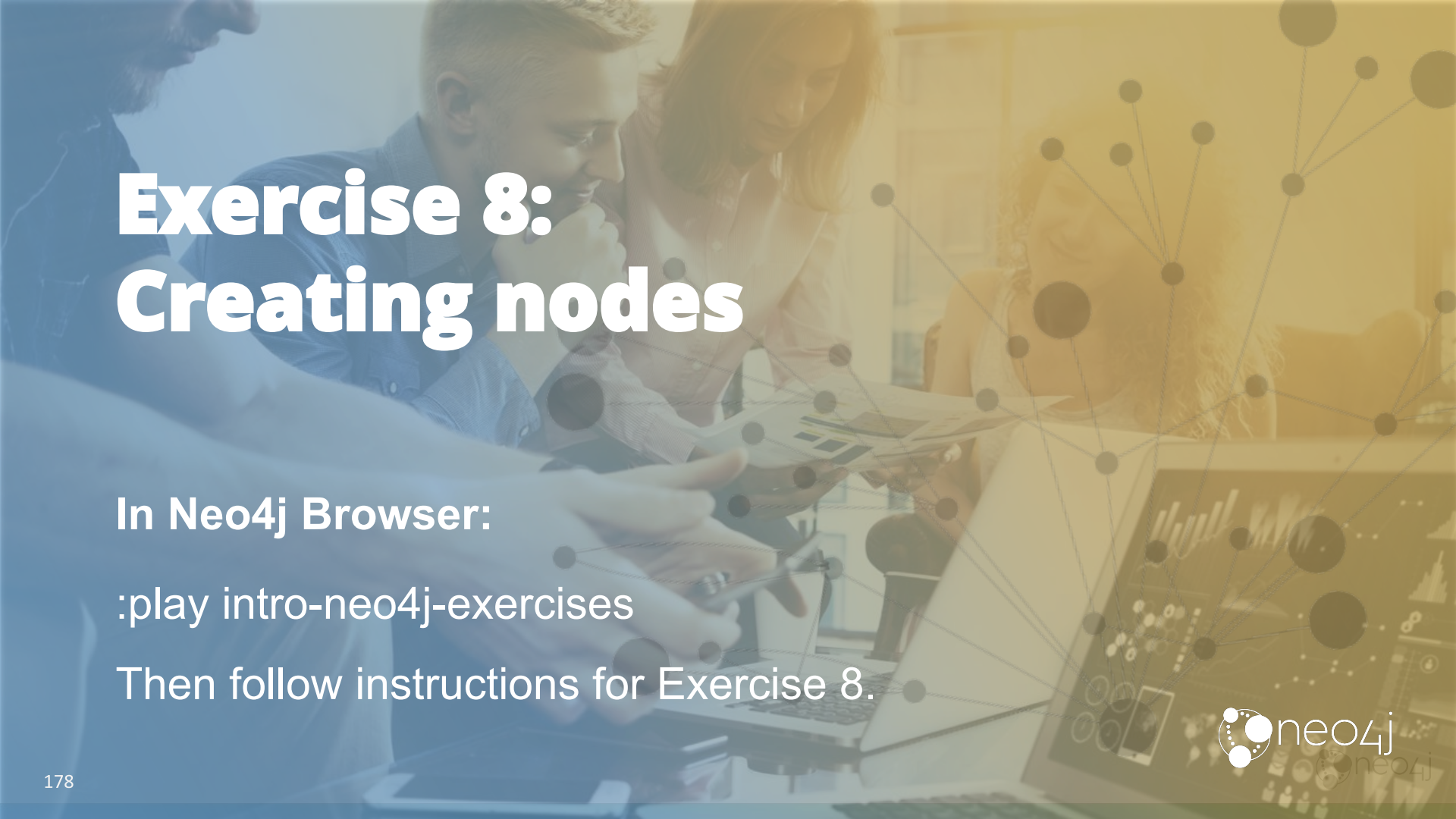
Properties can be removed in one of two ways:

- Set the property value to null
- Use the REMOVE keyword

Remove the grossMillions and videoFormat properties:

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





Exercise 8: Creating nodes

In Neo4j Browser:

:play intro-neo4j-exercises

Then follow instructions for Exercise 8.

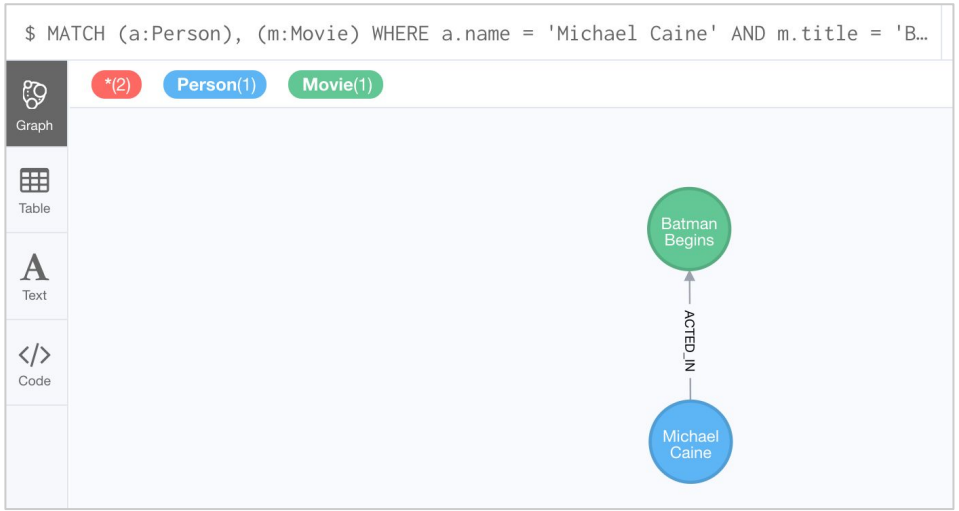
Creating a relationship

You create a relationship by:

1. Finding the “from node”.
2. Finding the “to node”.
3. Using CREATE to add the directed relationship between the nodes.

Create the *:ACTED_IN* relationship between the *Person*, *Michael Caine* and the *Movie*, *Batman Begins*:

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

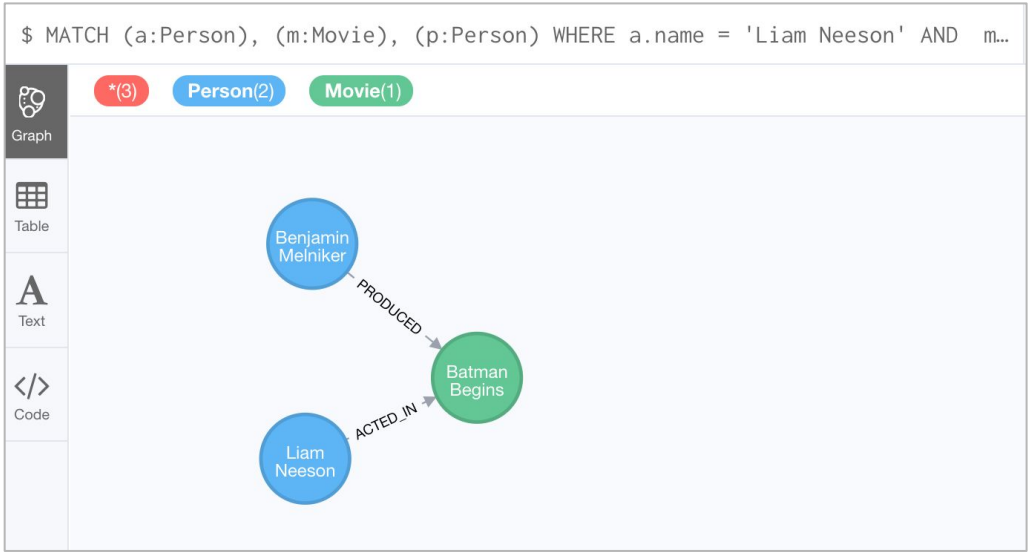


neo4j

Creating multiple relationships

Create the *:ACTED_IN* relationship between the *Person*, *Liam Neeson* and the *Movie*, *Batman Begins and the :PRODUCED* relationship between the *Person*, *Benjamin Melniker* and same movie.

```
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
```



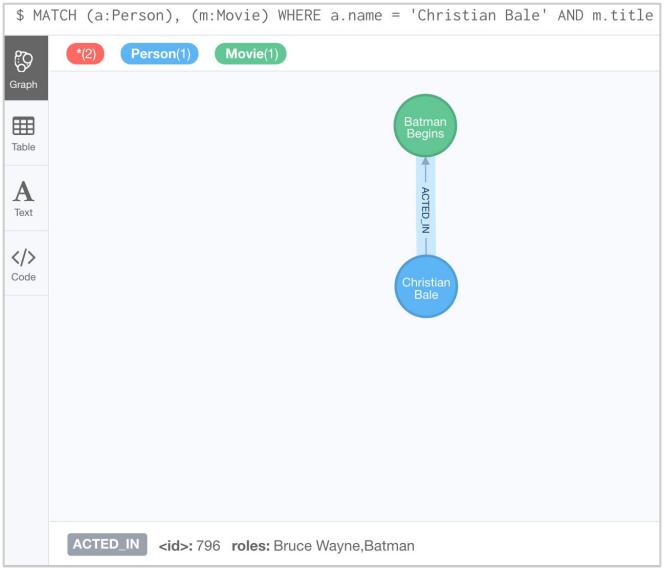
neo4j

Adding properties to relationships

Same technique you use for creating and updating node properties.

Add the *roles* property to the *:ACTED_IN* relationship from Christian Bale to *Batman Begins*:

```
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, m
```



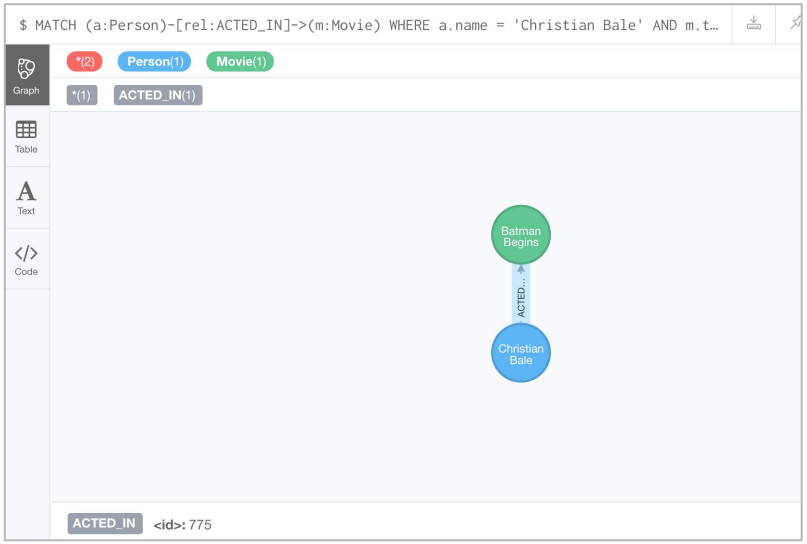
neo4j

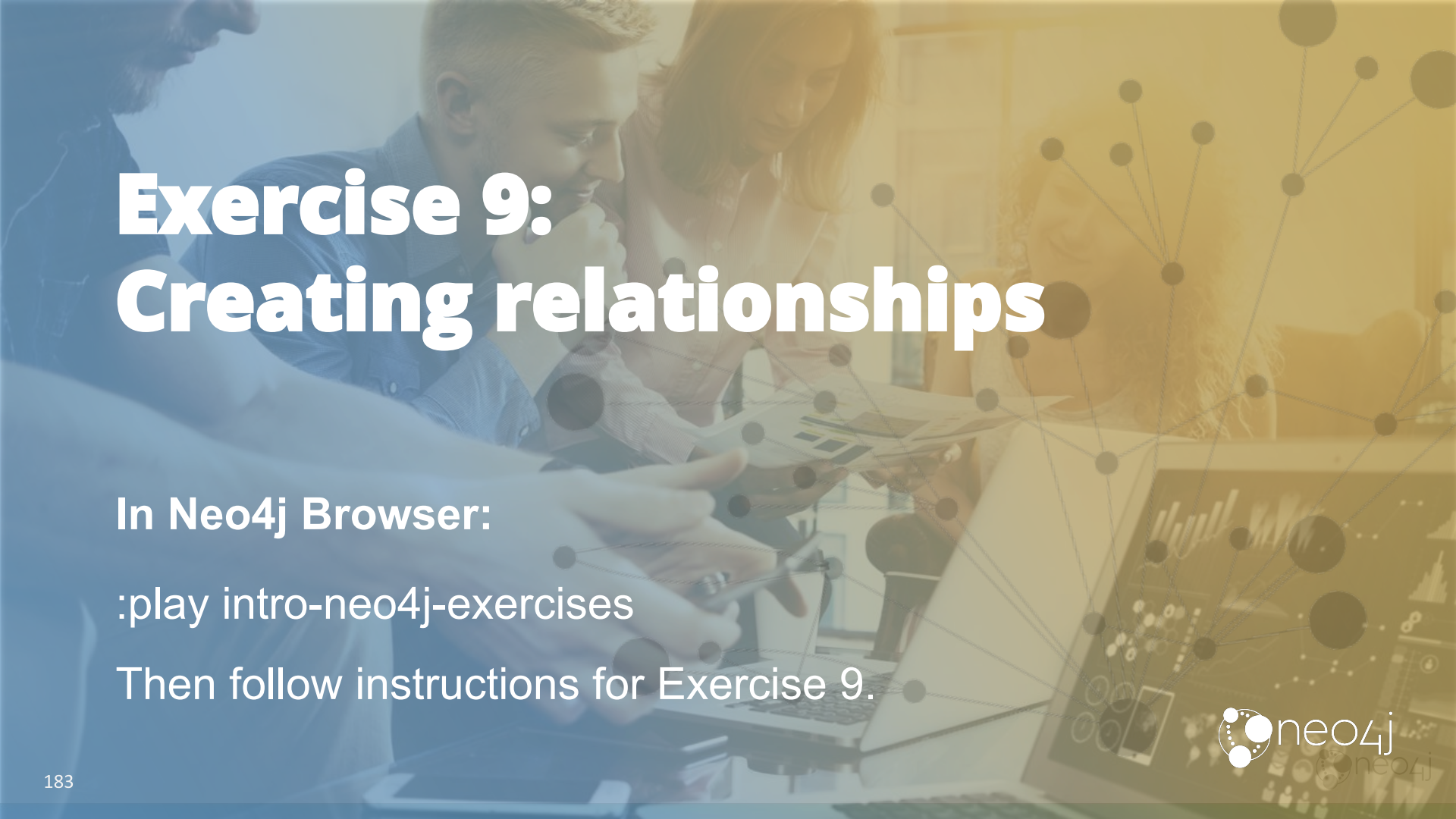
Removing properties from relationships

Same technique you use for removing node properties.

Remove the *roles* property from the *:ACTED_IN* relationship from Christian Bale to *Batman Begins*:

```
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
```





Exercise 9: Creating relationships

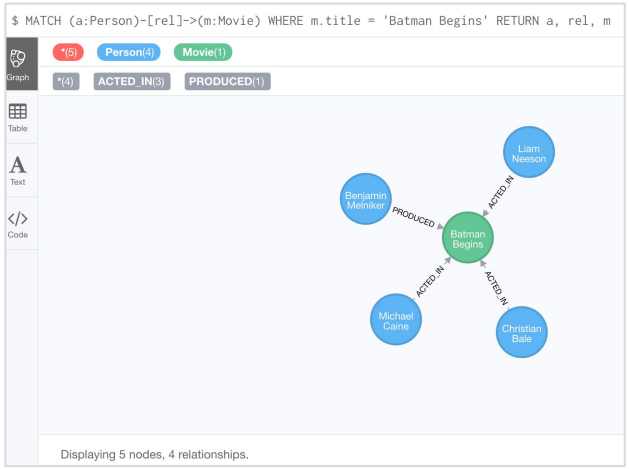
In Neo4j Browser:

:play intro-neo4j-exercises

Then follow instructions for Exercise 9.

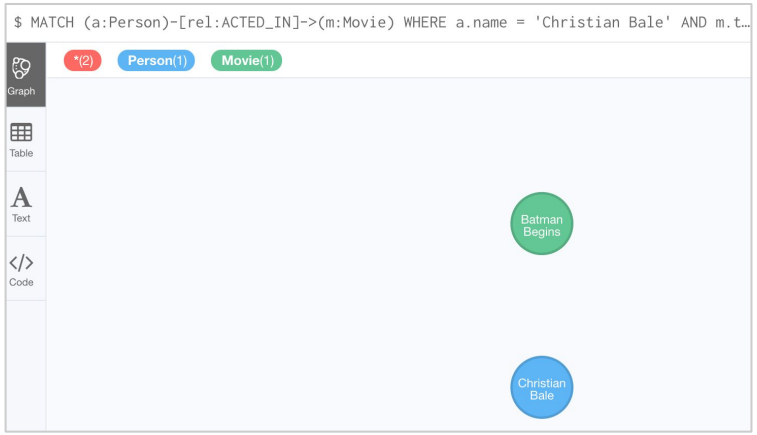
Deleting a relationship

Batman Begins relationships:



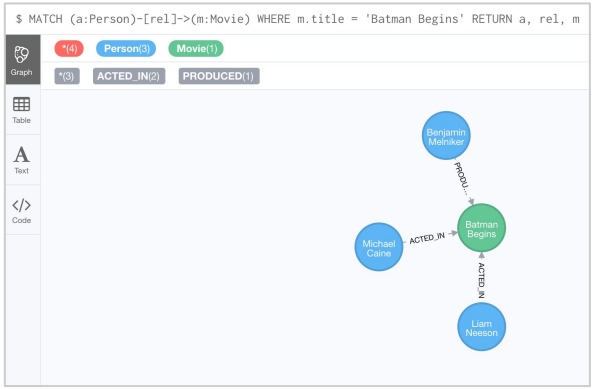
Delete the *:ACTED_IN* relationship between *Christian Bale* and *Batman Begins*:

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

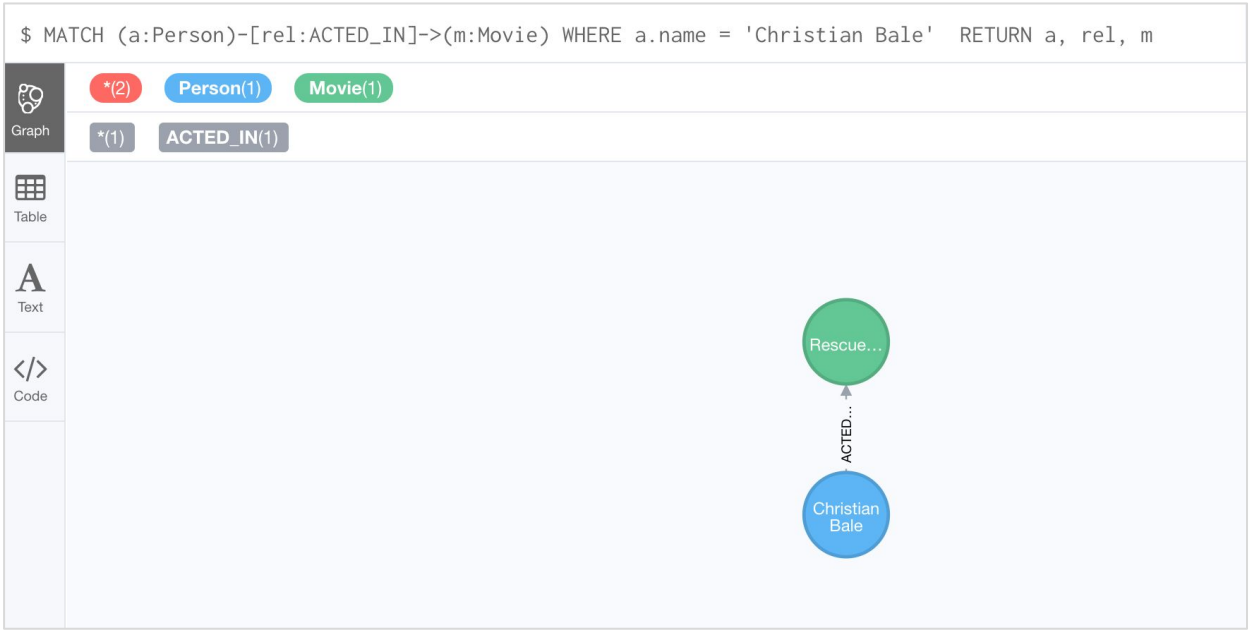


After deleting the relationship from *Christian Bale* to *Batman Begins*

Batman Begins relationships:



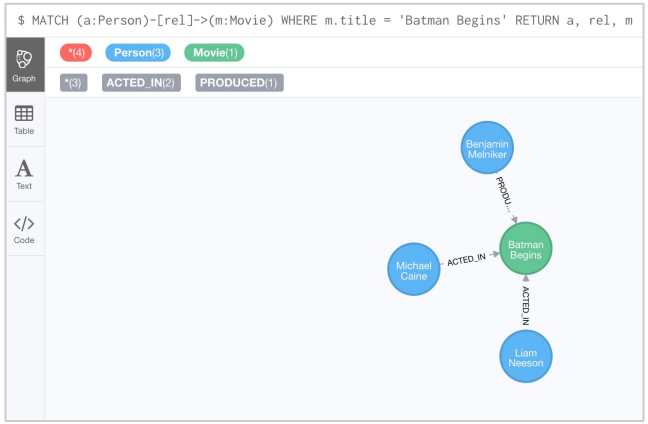
Christian Bale relationships:



neo4j

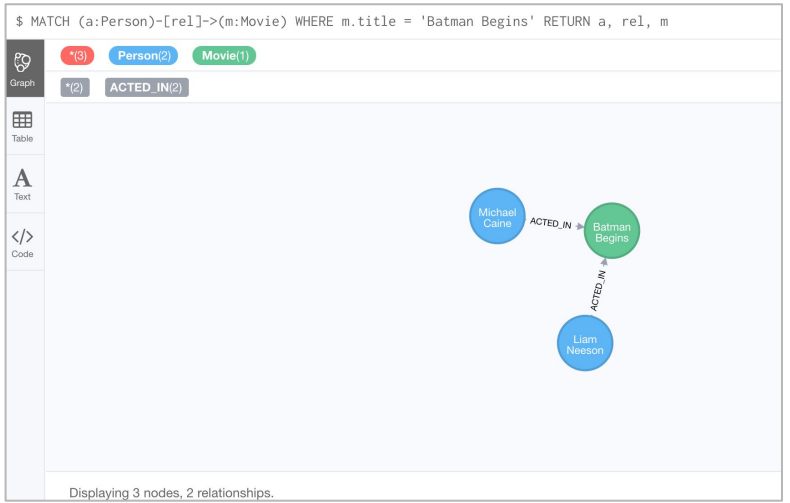
Deleting a relationship and a node - 1

Batman Begins relationships:



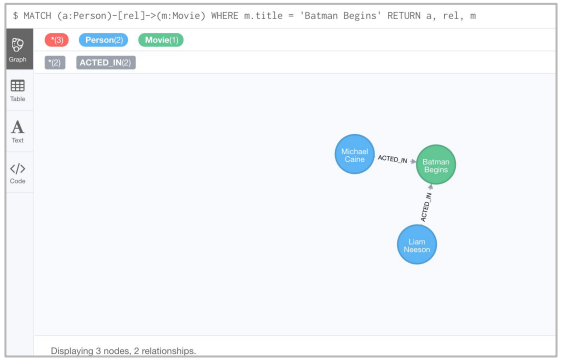
Delete the *:PRODUCED* relationship between *Benjamin Melniker* and *Batman Begins,* as well as the *Benjamin Melniker* node:

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



Deleting a relationship and a node - 2

Batman Begins relationships:



Attempt to delete *Liam Neeson* and not his relationships to any other nodes:

```
MATCH (p:Person)
WHERE p.name = 'Liam Neeson'
DELETE p
```

```
$ MATCH (p:Person) WHERE p.name = 'Liam Neeson' DELETE p
```

Error

ERROR

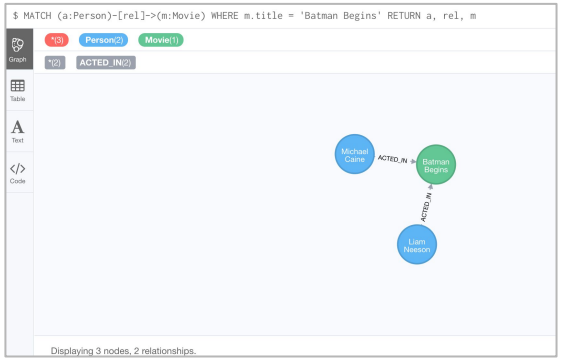
Neo.ClientError.Schema.ConstraintValidationFailed

```
Neo.ClientError.Schema.ConstraintValidationFailed: Cannot delete node<1899>, because it still
has relationships. To delete this node, you must first delete its relationships.
```

⚠ Neo.ClientError.Schema.ConstraintValidationFailed: Cannot delete node<1899>, because it still has relationships. To delete this node, you must first…

Deleting a relationship and a node - 3

Batman Begins relationships:



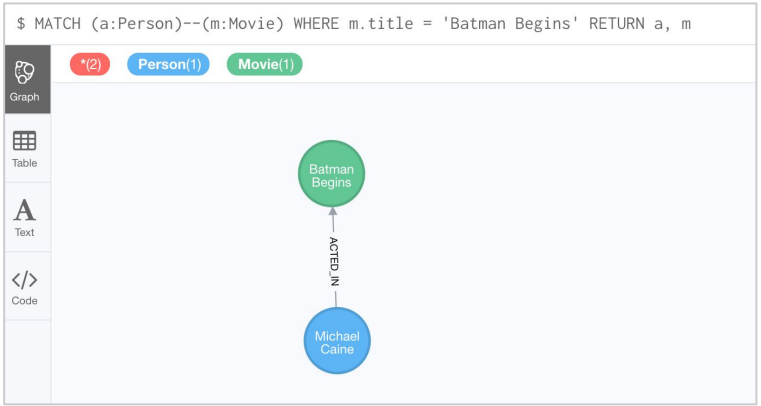
Delete *Liam Neeson* and his relationships to any other nodes:

```
MATCH (p:Person)
WHERE p.name = 'Liam Neeson'
DETACH DELETE p
```

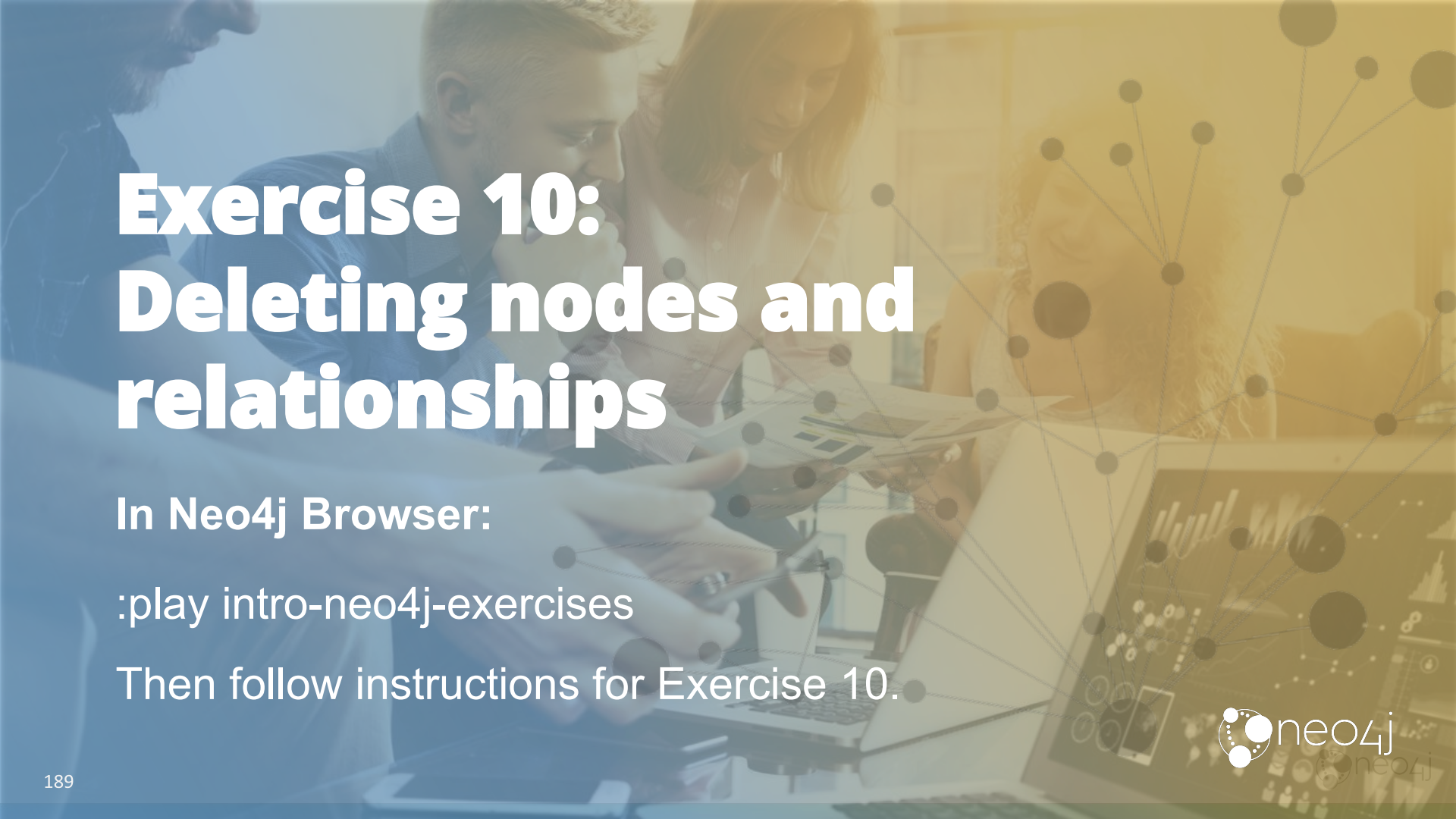
```
$ MATCH (p:Person) WHERE p.name = 'Liam Neeson' DETACH DELETE p
```

Table

Deleted 1 node, deleted 1 relationship, completed after 10 ms.



neo4j



Exercise 10: Deleting nodes and relationships

In Neo4j Browser:

:play intro-neo4j-exercises

Then follow instructions for Exercise 10.

Merging data in a graph

- 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

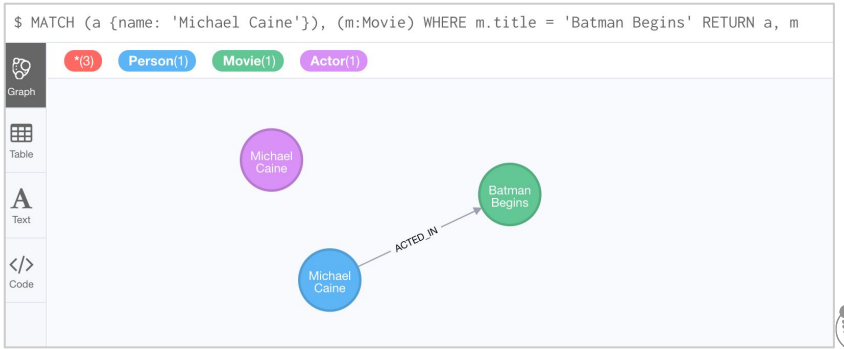
Current *Michael Caine Person* node:

Add a *Michael Caine Actor* node with a value of *1933* for *born* using MERGE. The *Actor* node is not found so a new node is created:

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

Important: Only specify properties that will have unique keys when you merge.

Resulting *Michael Caine nodes*:



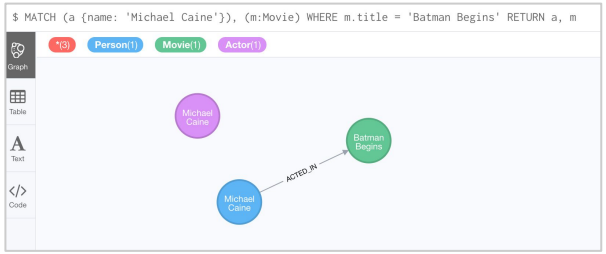
Using MERGE to create relationships

Add the relationship(s) from all *Person* nodes with a *name* property that ends with *Caine* to the *Movie* node, *Batman Begins*:

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

Specifying creation behavior for the merge

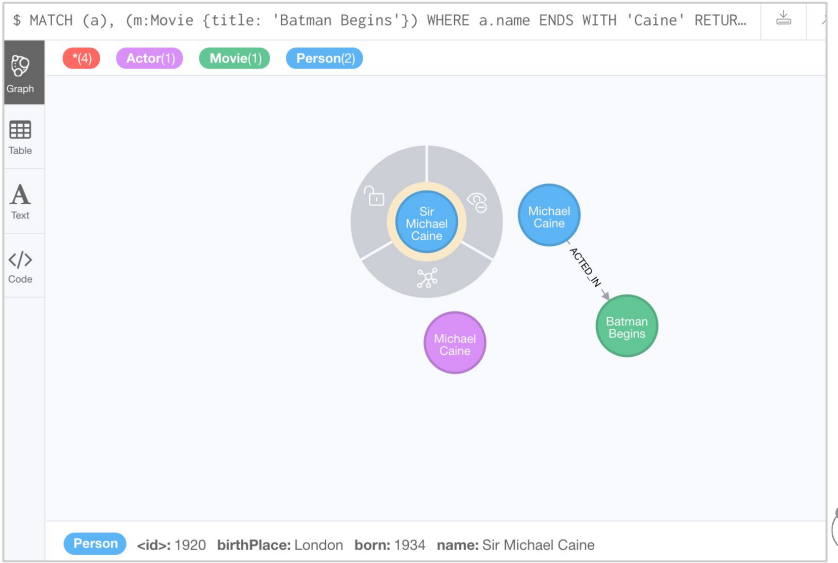
Current *Michael Caine* nodes:



Add a *Sir Michael Caine Person* node with a *born* value of *1934* for *born* using MERGE and also set the *birthPlace* property:

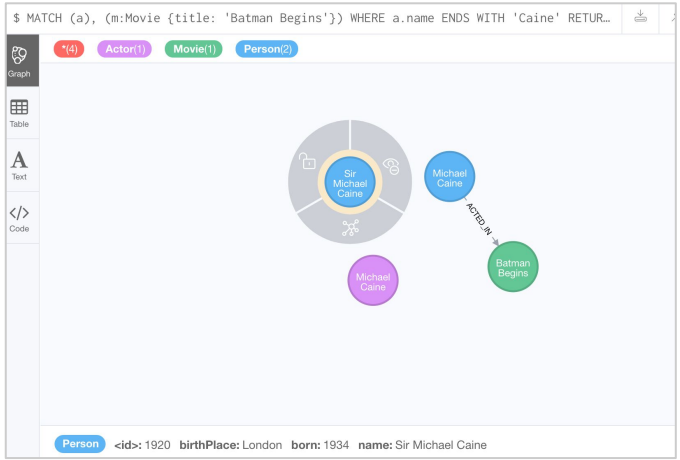
```
MERGE (a:Person {name: 'Sir Michael Caine'})
ON CREATE SET a.born = 1934,
              a.birthPlace = 'London'
RETURN a
```

Resulting *Michael Caine nodes*:



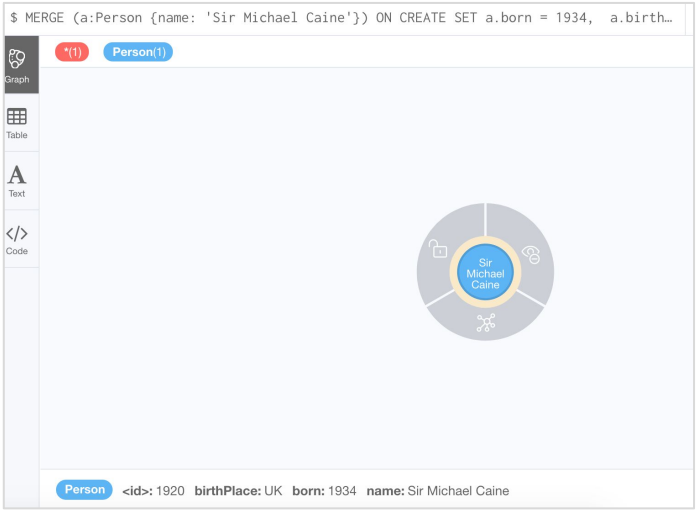
Specifying match behavior for the merge

Current *Michael Caine* nodes:



Add or update the *Michael Caine Person* node:

```
MERGE (a:Person {name: 'Sir Michael Caine'})
ON CREATE SET a.born = 1934,
              a.birthPlace = 'UK'
ON MATCH SET a.birthPlace = 'UK'
RETURN a
```

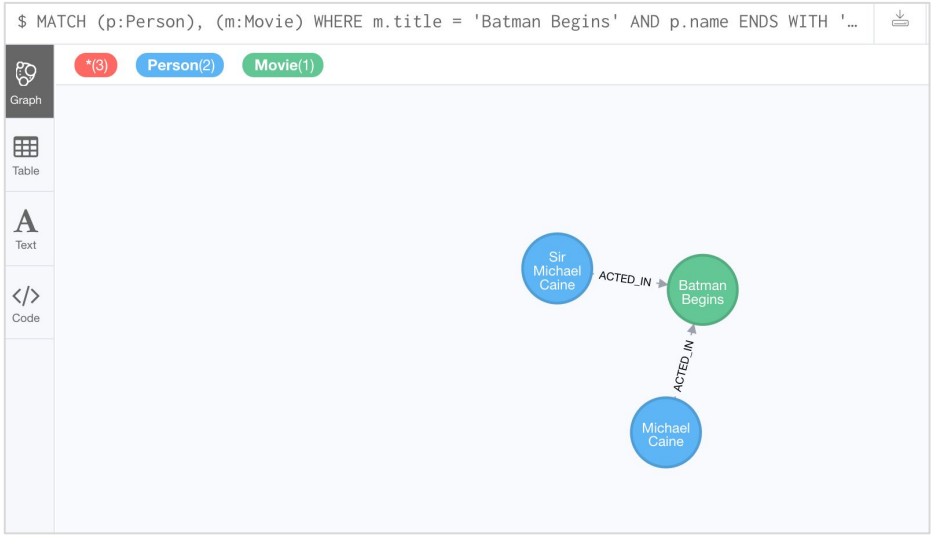


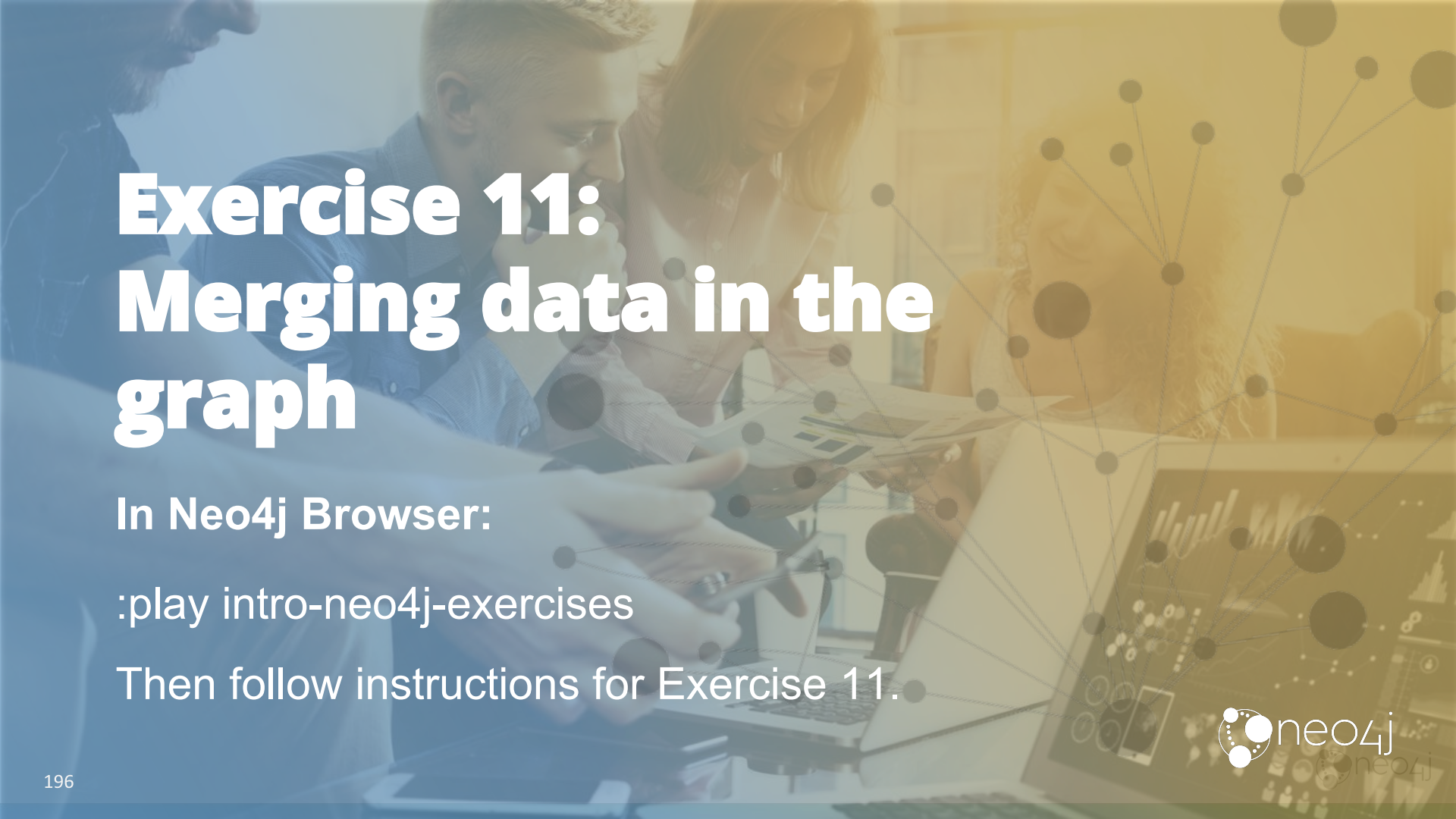
neo4j

Using MERGE to create relationships

Make sure that all *Person* nodes with a person whose name ends with *Caine* are connected to the *Movie* node, *Batman Begins*.

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





Exercise 11: Merging data in the graph

In Neo4j Browser:

:play intro-neo4j-exercises

Then follow 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

Answer 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, *s* with a property, *color*:

What Cypher clause 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=?

Answer 2

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

What Cypher clause 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 clause 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

Answer 3

Suppose you retrieve a node, *n* in the graph that is related to other nodes. What Cypher clause 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 be able to write Cypher statements 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.


Getting More Out of Neo4j
v 1.0
neo4j

Overview

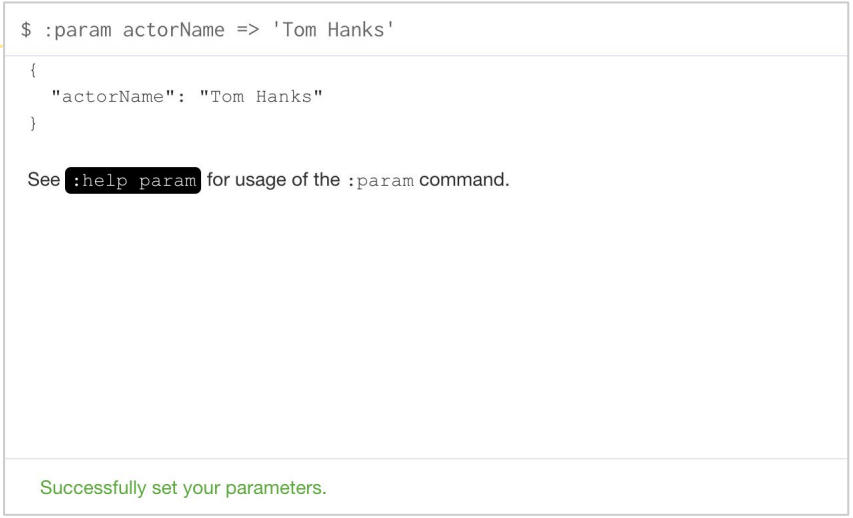
At the end of this module, you should be able to:

- Use parameters in your Cypher statements.
- Analyze Cypher execution.
- Monitor queries.
- Manage constraints and node keys for the graph.
- Import data into a graph from CSV files.
- Manage indexes for the graph.
- Access Neo4j resources.

Cypher parameters

Using Cypher parameters - 1

1. Set values for parameters in your Neo4j Browser session before you run the query.



2. Specify parameters using '$' in your Cypher query.

```
MATCH (p:Person)-[:ACTED_IN]->(m:Movie)
WHERE p.name = $actorName
RETURN m.released, m.title ORDER BY m.released DESC
```

Using Cypher parameters - 2

When this query runs, *$actorName* has a value *Tom Hanks*:

```
$ MATCH (p:Person)-[:ACTED_IN]->(m:Movie) WHERE p.name = $actorName RETURN m.released, m.title O…
```

Table | Text | Code

m.released	m.title
2012	"Cloud Atlas"
2007	"Charlie Wilson's War"
2006	"The Da Vinci Code"
2004	"The Polar Express"
2000	"Cast Away"
1999	"The Green Mile"
1998	"You've Got Mail"
1996	"That Thing You Do"
1995	"Apollo 13"
1994	"Forrest Gump"
1993	"Sleepless in Seattle"
1992	"A League of Their Own"
1990	"Joe Versus the Volcano"

Using Cypher parameters - 3

Change the value of the parameter, *$actorName* to *Tom Cruise*:

```
:param actorName => 'Tom Cruise'
```

Re-run the same query:

```
$ MATCH (p:Person)-[:ACTED_IN]->(m:Movie) WHERE p.name = $actorName RETURN m.released, m.title O…
```

Table

m.released	m.title
2000	"Jerry Maguire"
1992	"A Few Good Men"
1986	"Top Gun"

Text

</>

Setting and viewing Cypher parameters - JSON style

```
$ :params {actorName: 'Tom Cruise', movieName: 'Top Gun'}
```

```
{
  "actorName": "Tom Cruise",
  "movieName": "Top Gun"
}
```

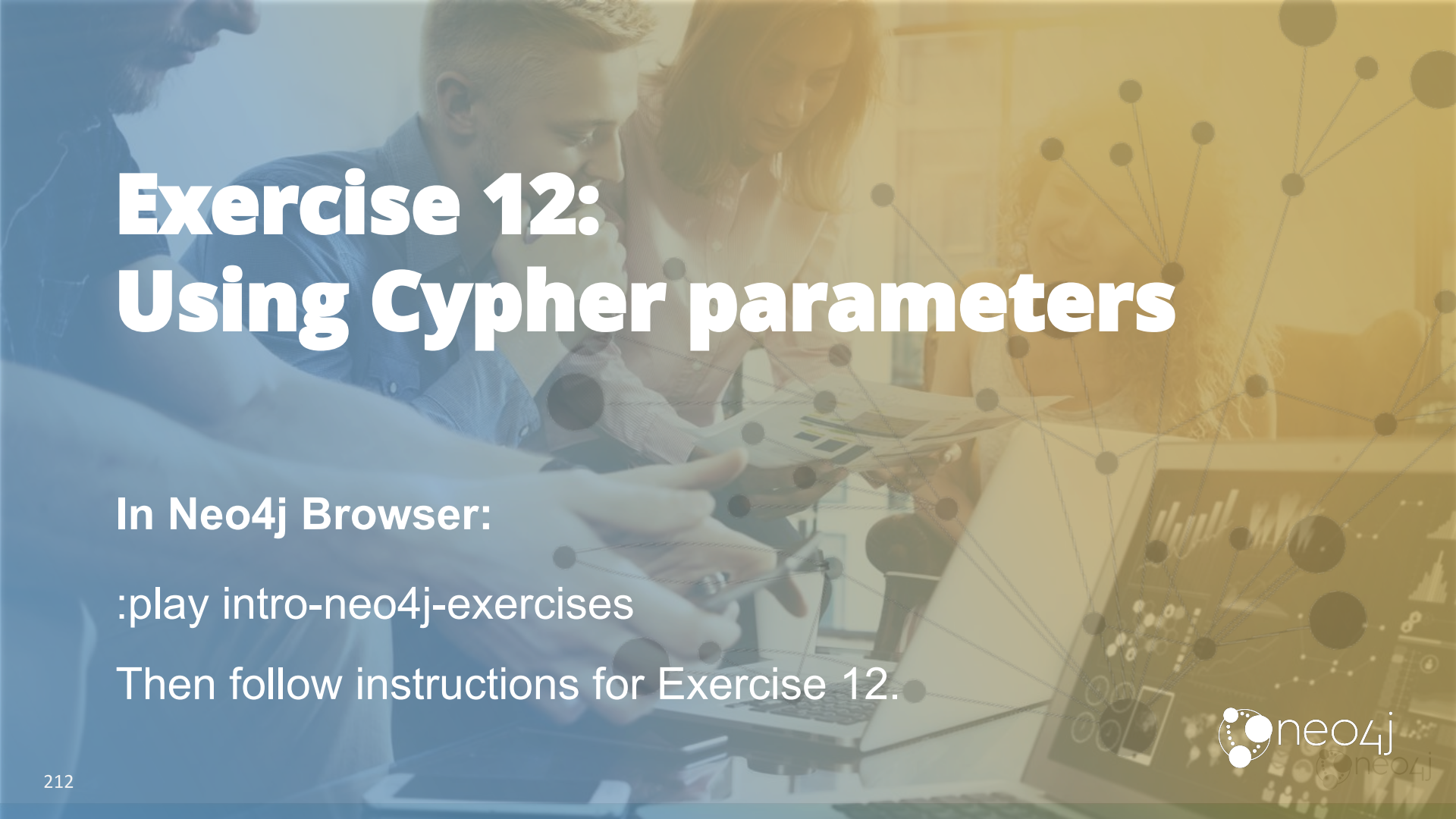
See `:help param` for usage of the `:param` command.

Successfully set your parameters.

```
$ :params
```

```
{
  "actorName": "Tom Cruise",
  "movieName": "Top Gun"
}
```

See `:help param` for usage of the `:param` command.



Exercise 12: Using Cypher parameters

In Neo4j Browser:

:play intro-neo4j-exercises

Then follow instructions for Exercise 12.

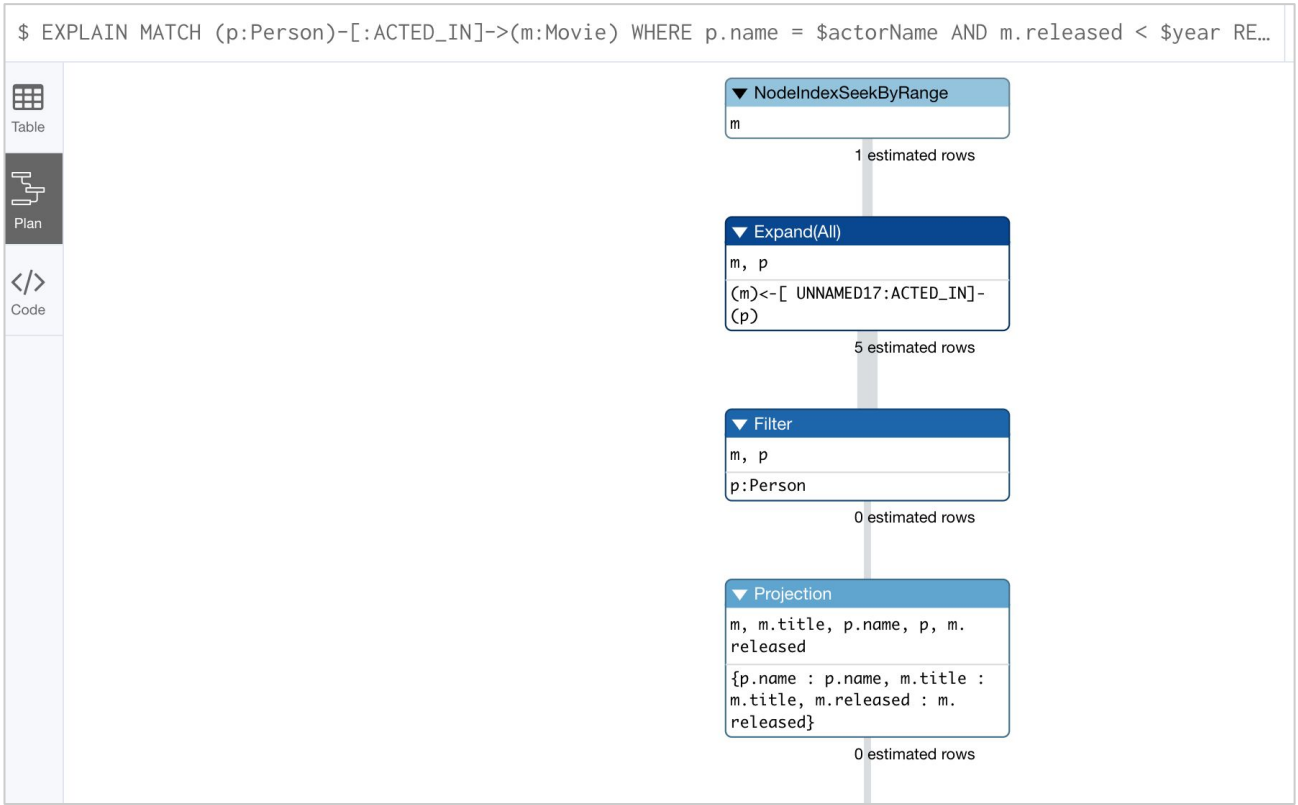
Analyzing Cypher queries - EXPLAIN - 1

- Provides information about the query plan.
- Does not execute the Cypher statement.

Here is an example where we have set the *$actorName* and *$year* parameters for our session and we execute this Cypher statement to produce the query plan:

```
EXPLAIN MATCH (p:Person)-[:ACTED_IN]->(m:Movie)
WHERE p.name = $actorName AND
      m.released <  $year
RETURN p.name, m.title, m.released
```

Analyzing Cypher queries - EXPLAIN - 2



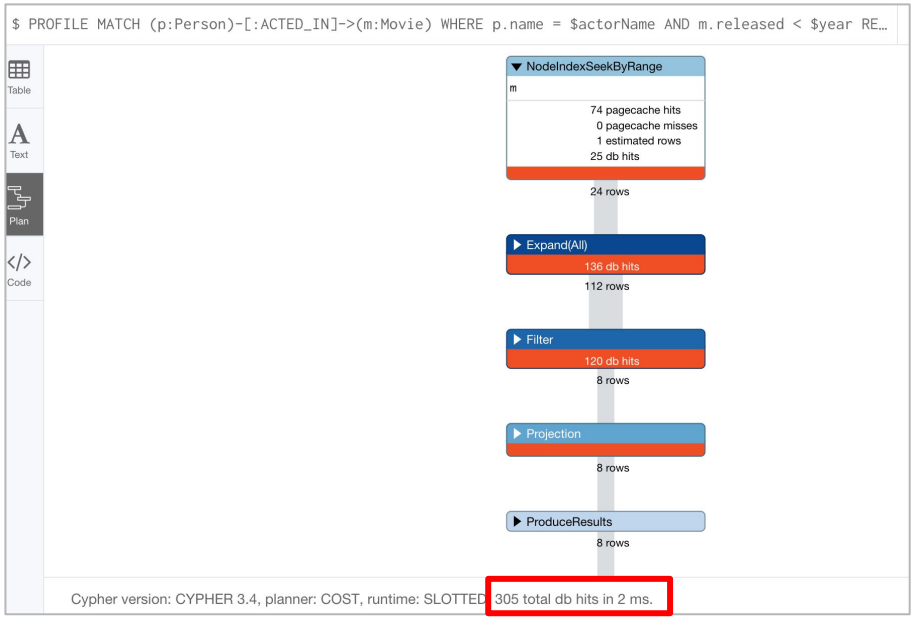
Analyzing Cypher queries - PROFILE - 1

- Provides information about the query plan.
- Executes the Cypher statement.
- Provides information about db hits.

```
PROFILE MATCH (p:Person)-[:ACTED_IN]->(m:Movie)
WHERE p.name = $actorName AND
      m.released <  $year
RETURN p.name, m.title, m.released
```

Analyzing Cypher queries - PROFILE - 2

Profile query where node labels are specified:



Profile query where node labels <u>not</u> are specified:

Monitoring queries

There are two reasons why a Cypher query may take a long time:

1. The query returns a lot of data. The query completes execution in the graph engine, but it takes a long time to create the result stream to return to the client.

```
MATCH (a)--(b)--(c)--(d)--(e)--(f) RETURN a
```

You should avoid these type of queries! You cannot monitor them.

2. The query takes a long time to execute in the graph engine.

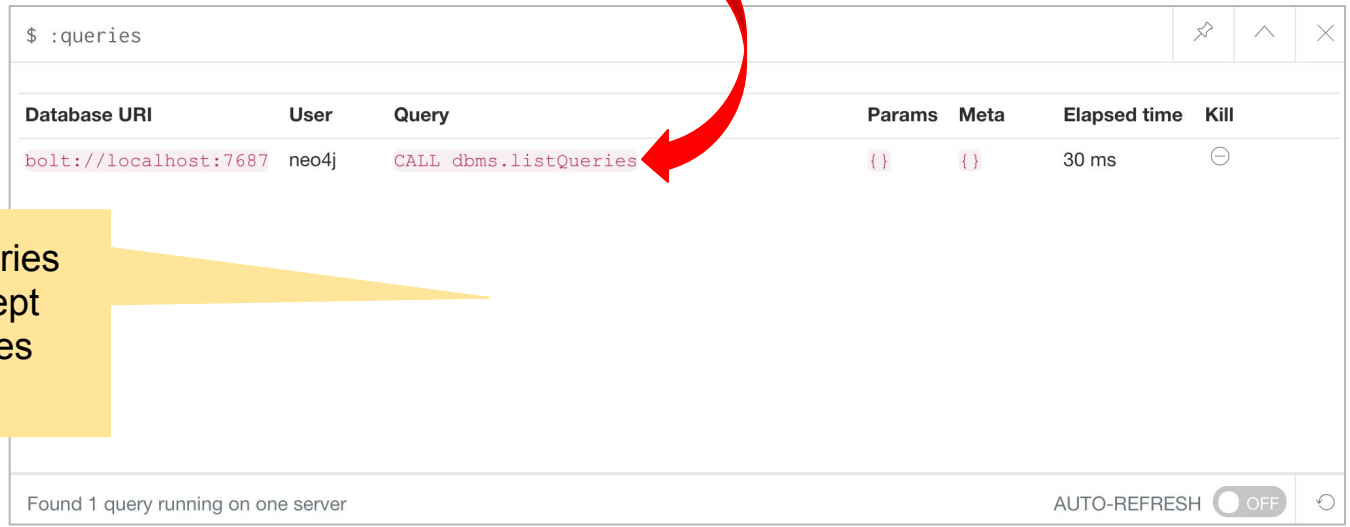
```
MATCH (a), (b), (c), (d), (e) RETURN count(id(a))
```

You can monitor and kill these types of queries.

Viewing running queries

If your query is taking a long time to execute your first have to determine if it is running in the graph engine:

1. Open a new Neo4j Browser session.
2. Execute the **:queries** command.



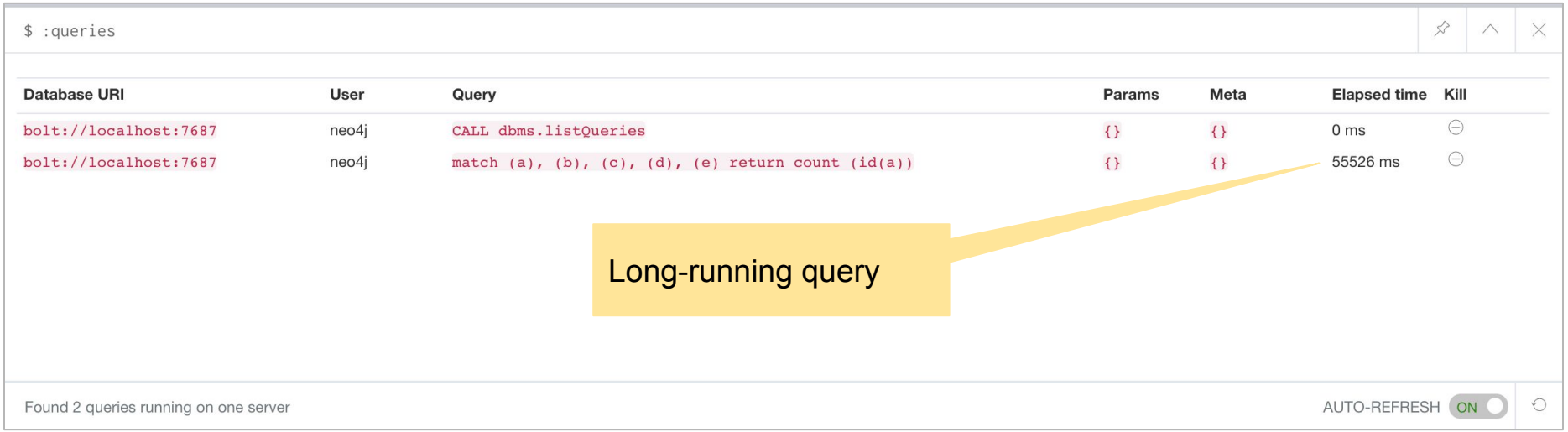
No other queries running, except for the :queries command.

Handling “rogue” queries

If your query is taking a long time to execute and you cannot monitor it, your options are to:

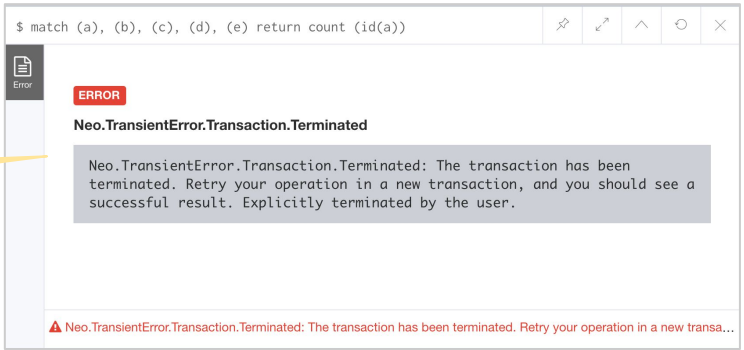
1. Close the Neo4j Browser session that is stuck and start a new Neo4j Browser session.
2. If that doesn’t work:
   a. On Neo4j Desktop, restart the database.
   b. In Neo4j Sandbox, shut down the sandbox (ouch!). You need to re-create the Sandbox.

Viewing long-running queries

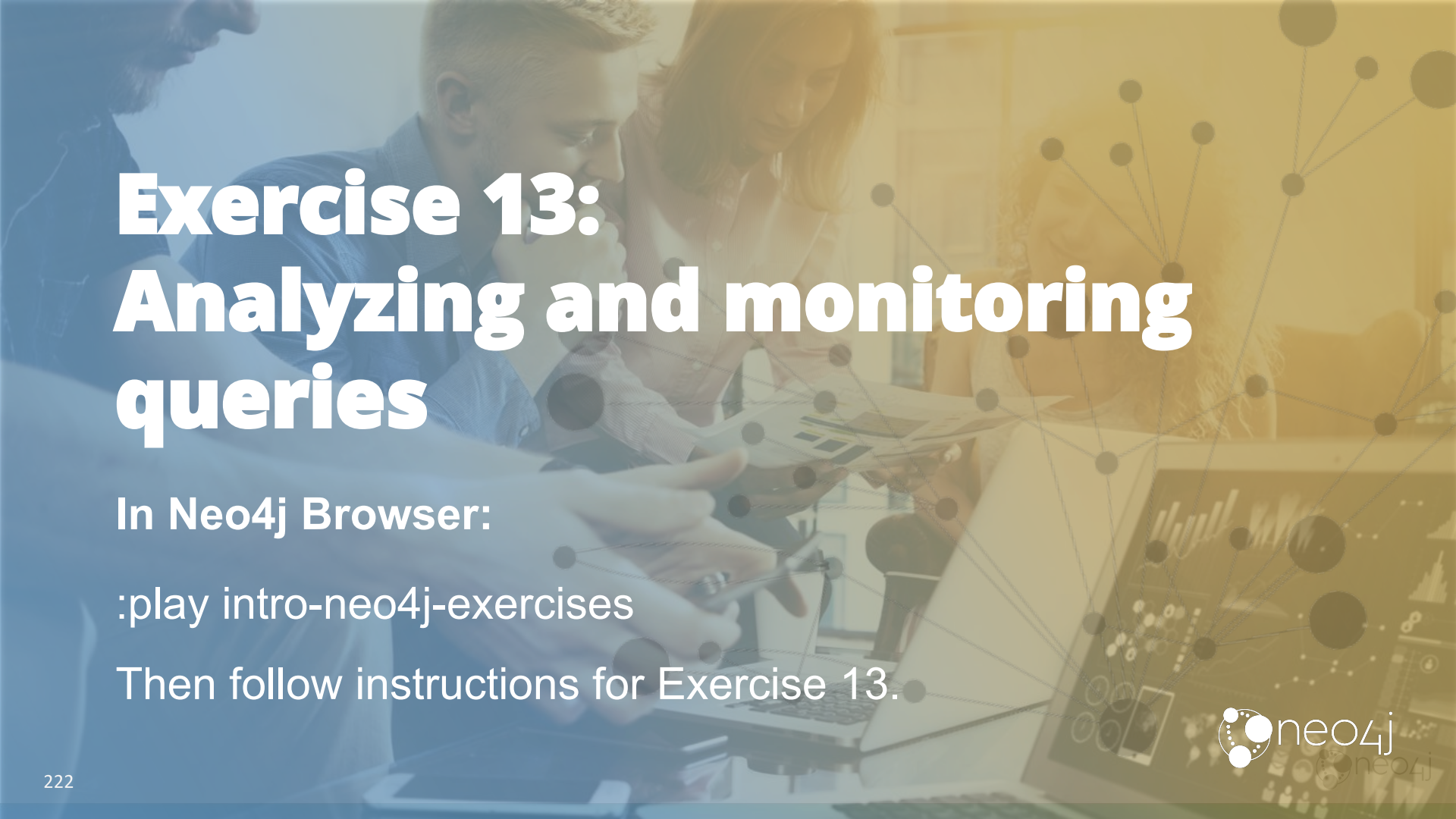


Killing long-running queries

Monitoring session



Query session



Exercise 13: Analyzing and monitoring queries

In Neo4j Browser:

:play intro-neo4j-exercises

Then follow instructions for Exercise 13.

Managing constraints and node keys

Automatically control the data that is added to the graph:

- **Uniqueness:** Unique values for node properties
- **Existence**: Required properties for nodes or relationships

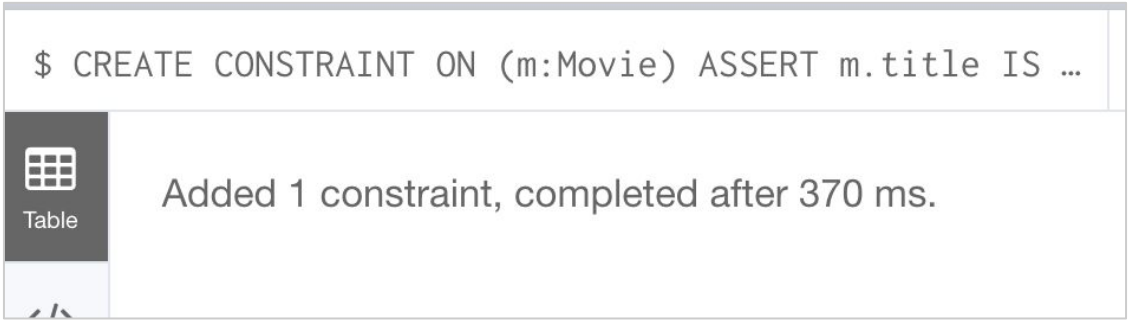
Note:

- You can write additional code in your Cypher statements to assert values, but it is much easier to let the graph engine do it for you.
- You can also write server-side extensions which are beyond the scope of this training.

Ensuring that a property value for a node is unique

Ensure that the *title* for a node of type *Movie* is unique:

```
CREATE CONSTRAINT ON (m:Movie) ASSERT m.title IS UNIQUE
```

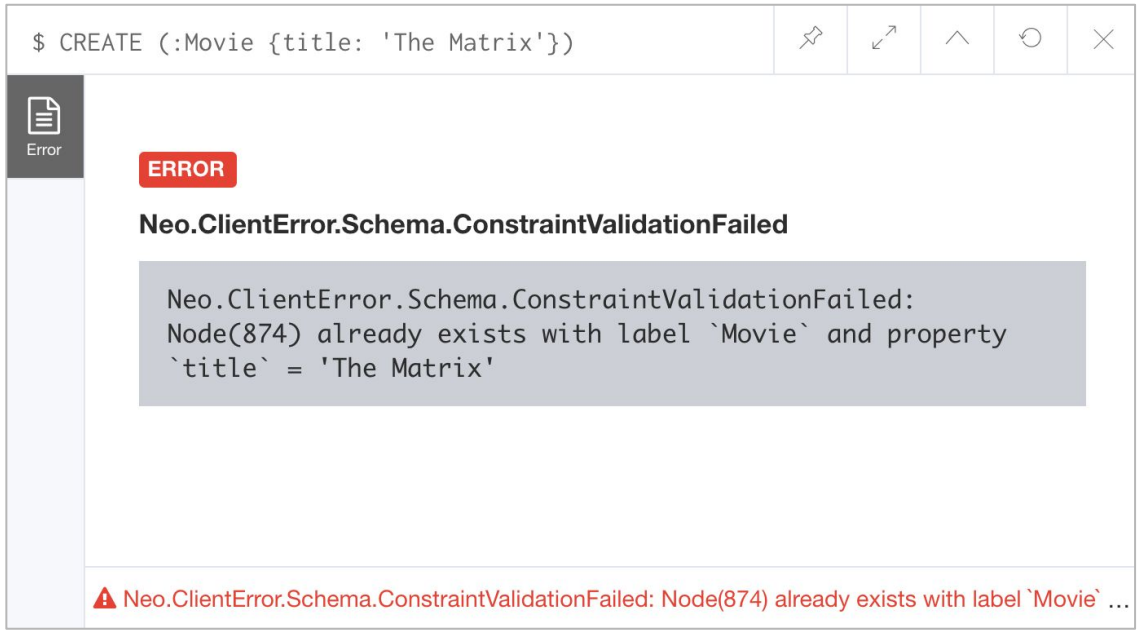


- This statement will fail if there are any *Movie* nodes in the graph that have the same value for the *title* property.
- This statement will succeed if there are any *Movie* nodes in the graph that do not have the *title* property.

Ensuring uniqueness using the constraint

After creating the constraint, we attempt to create a *Movie* with the *title*, The *Matrix*:

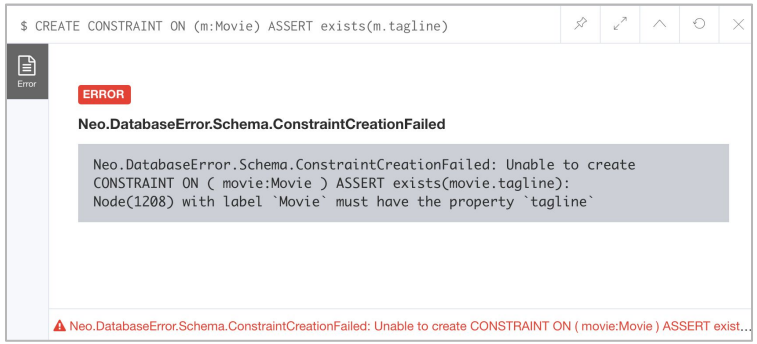
```
CREATE (:Movie {title: 'The Matrix'})
```



Ensuring that properties exist

You can create an constraint that will ensure that when a node or relationship is created or updated, a particular property must have a value:

```
CREATE CONSTRAINT ON (m:Movie) ASSERT exists(m.tagline)
```



This statement failed because the *Movie* node for the movie, *Something's Gotta Give* does not have a value for the *tagline* property.

Creating an exists constraint on a relationship

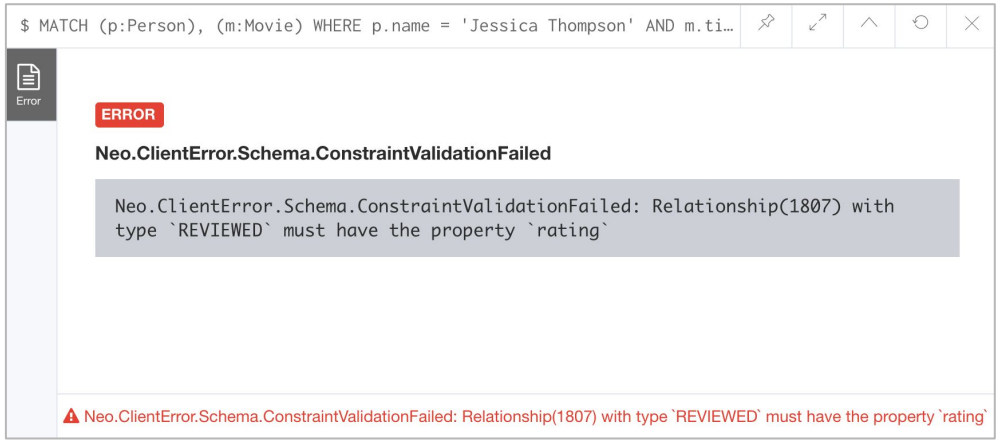
We know that in the *Movie* graph, all :*REVIEWED* relationships currently have a property, *rating*. We can create an existence constraint on that property as follows:

```
CREATE CONSTRAINT ON ()-[rel:REVIEWED]-() ASSERT exists(rel.rating)
```

Using the exists constraint on a relationship

After creating this constraint, if we attempt to create a *:REVIEWED* relationship without setting the *rating* property:

```
MATCH (p:Person), (m:Movie)
WHERE p.name = 'Jessica Thompson' AND
      m.title = 'The Matrix'
MERGE (p)-[:REVIEWED {summary: 'Great movie!'}]->(m)
```



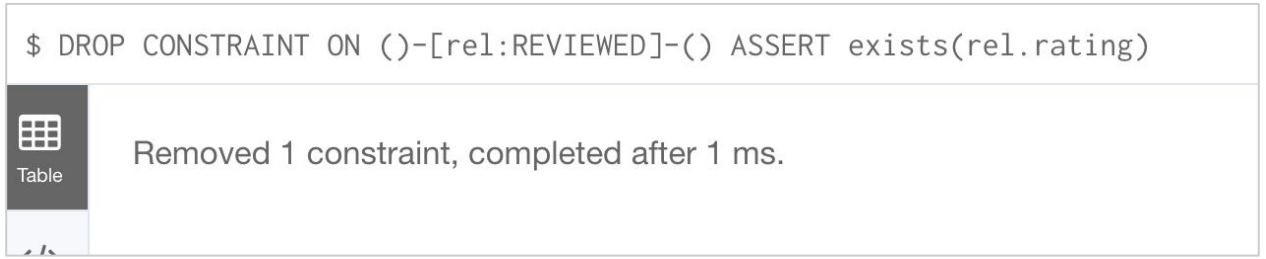
Retrieving constraints defined for the graph

```
CALL db.constraints()
```

Note: Adding the method notation for this CALL statement enables you to use the call for returning results that may be used later in the Cypher statement.

Dropping constraints

```
DROP CONSTRAINT ON ()-[rel:REVIEWED]-() ASSERT exists(rel.rating)
```



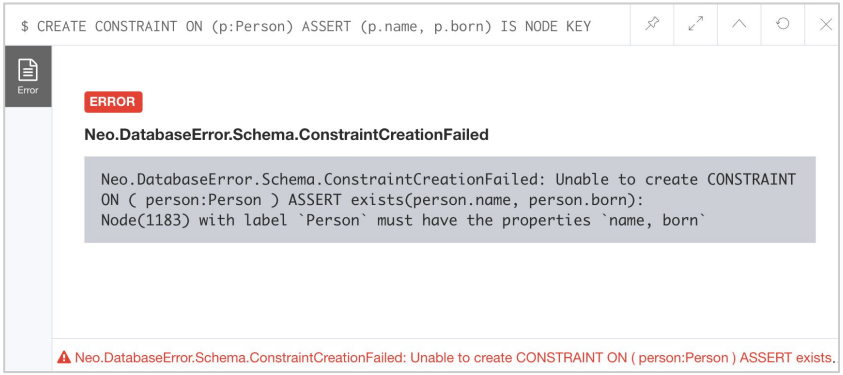
Creating node keys - 1

- Unique constraint for a set of properties for a node
- Is implemented as an index in the graph

Suppose that in our *Movie* graph, we will not allow a *Person* node to be created where both the *name* and *born* properties are the same. We can create a constraint that will be a node key to ensure that this uniqueness for the set of properties is asserted:

```
CREATE CONSTRAINT ON (p:Person) ASSERT (p.name, p.born) IS NODE KEY
```

We attempt to create the constraint, but it fails because there is a *Person* node in the graph that does not have the *born* property set:



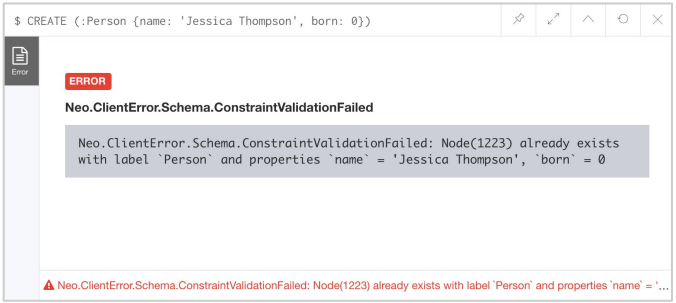
Creating node keys - 2

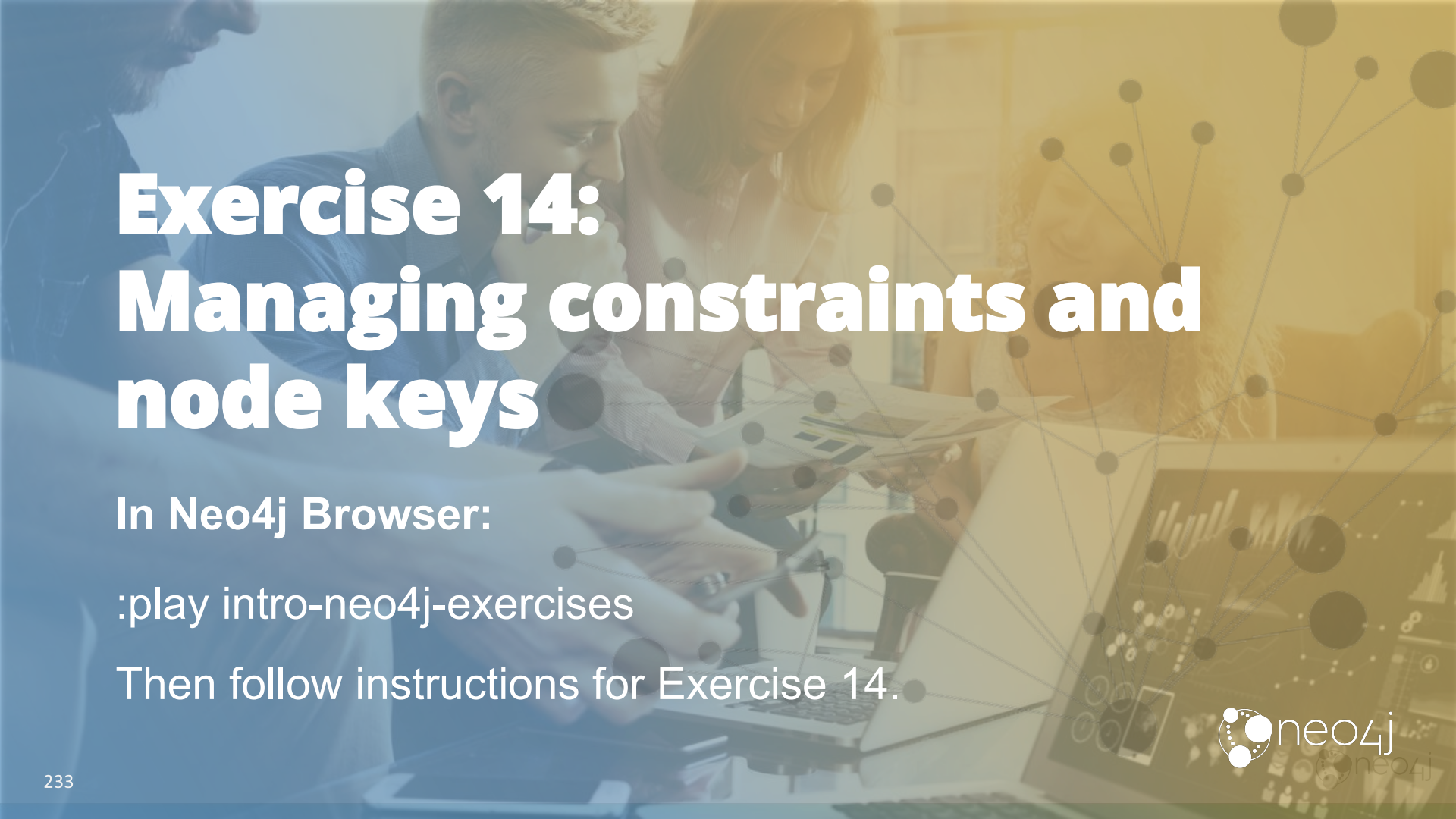
We then ensure that all *Person* nodes have a value for the *born* property:

```
MATCH (p:Person)
WHERE NOT exists(p.born)
SET p.born = 0
```

The creation of the node key will now be successful:

Any subsequent attempt to create or modify an existing *Person* node with *name* or *born* values that violate the uniqueness constraint as a node key will fail:





Exercise 14: Managing constraints and node keys

In Neo4j Browser:

:play intro-neo4j-exercises

Then follow instructions for Exercise 14.

Indexes in Neo4j

- Neo4j supports two types of indexes on a node of a specific type:
  - single property
  - composite properties
- Indexes store redundant data that points to nodes with the specific property value or values.
- Unlike SQL, there is no such thing as a primary key in Neo4j. You can have multiple properties on nodes that must be unique.
- Add indexes before you create relationships between nodes.
- Creating an index on a property does not guarantee uniqueness.
  - But uniqueness and node key constraints are indexes that guarantee uniqueness.

When indexes are used

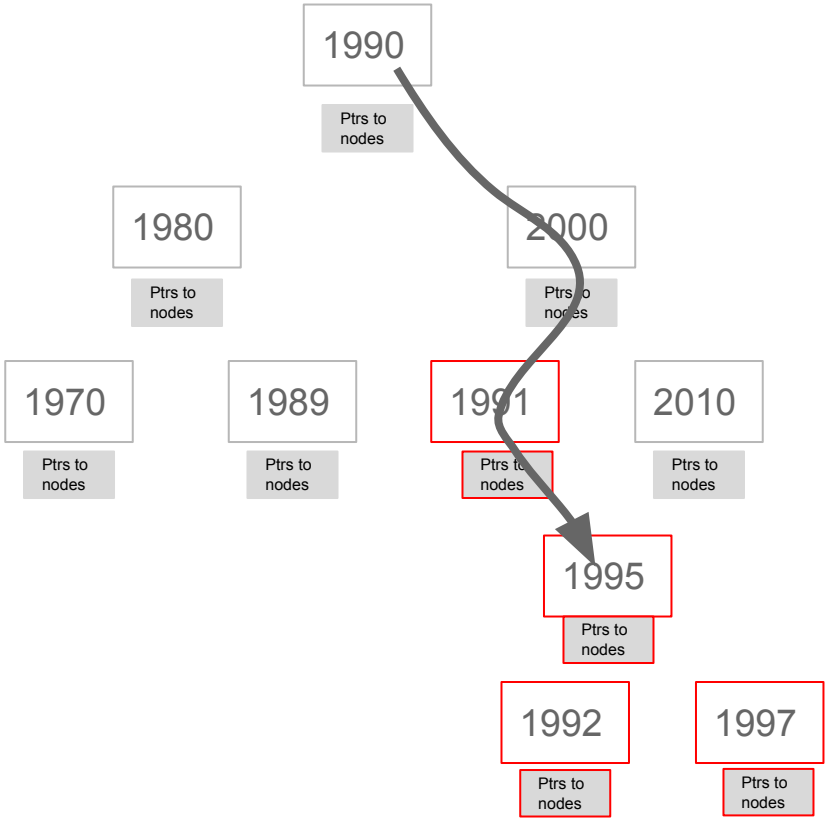
Single property indexes are used to determine the starting point for graph traversal using:

- Equality checks =
- Range comparisons >,>=,<, <=
- List membership IN
- String comparisons STARTS WITH, ENDS WITH, CONTAINS
- Existence checks exists()
- Spatial distance searches distance()
- Spatial bounding searches point()

Note: Composite indexes are only used for equality checks and list membership.

Indexes for range queries

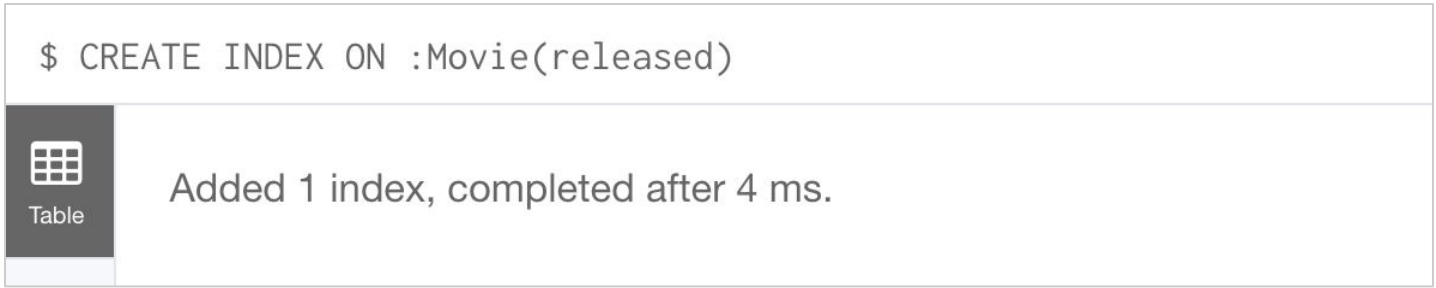
```
MATCH (m:Movie)
WHERE 1990 < m.released < 2000
SET m.videoFormat = 'DVD'
```



Creating a single-property index

Create a single-property index on the *released* property of all nodes of type *Movie*:

```
CREATE INDEX ON :Movie(released)
```



Creating a composite index - 1

Suppose first that we added the property, *videoFormat* to every *Movie* node and set its value, based upon the released date of the movie as follows:

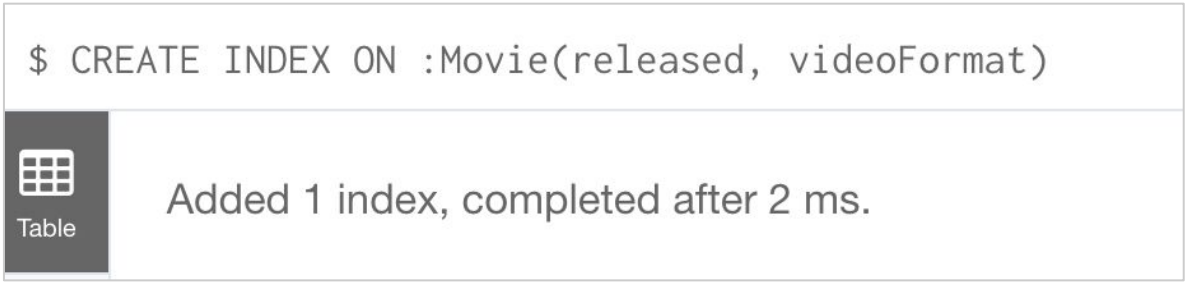
```
MATCH (m:Movie)
WHERE m.released >= 2000
SET m.videoFormat = 'DVD';
MATCH (m:Movie)
WHERE m.released < 2000
SET m.videoFormat = 'VHS'
```

All *Movie* nodes in the graph now have both a *released* and *videoFormat* property.

Creating a composite index - 2

Create a composite index for every *Movie* node that uses the *videoFormat* and *released* properties:

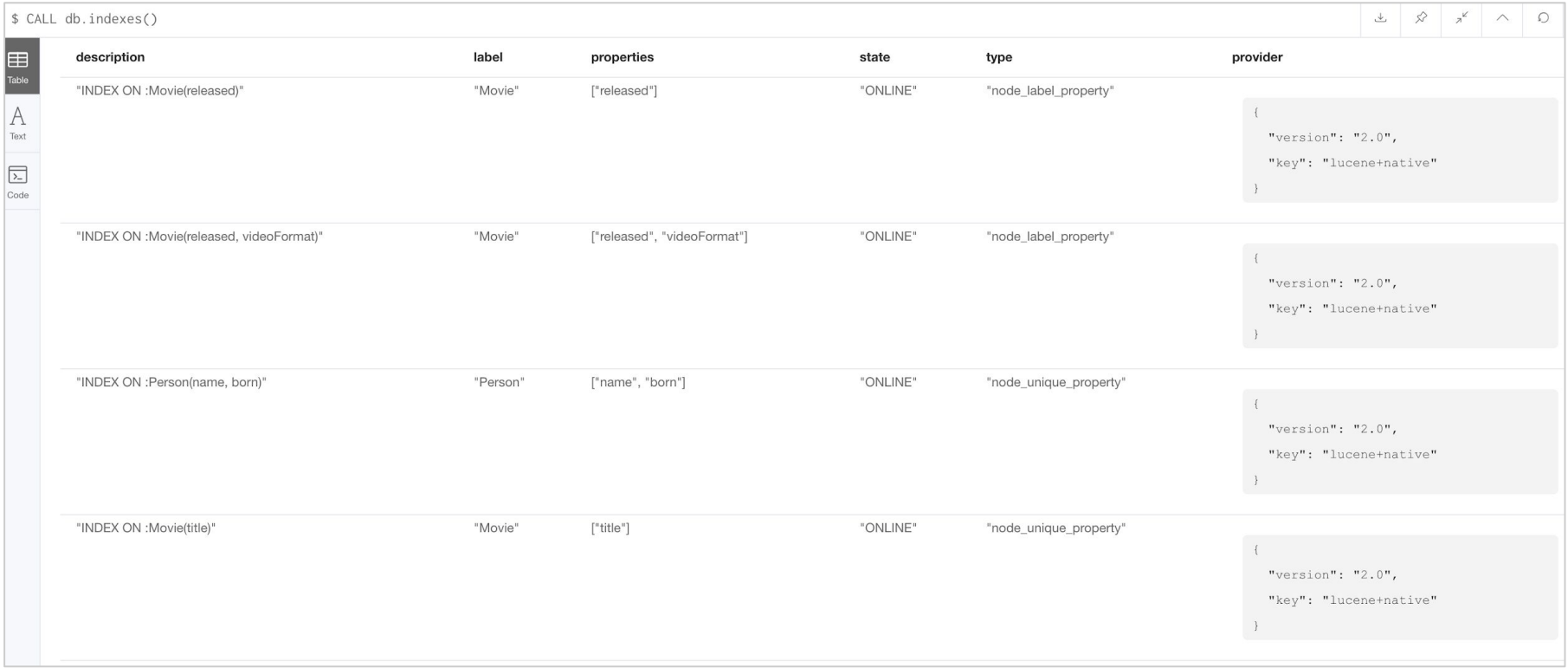
```
CREATE INDEX ON :Movie(released, videoFormat)
```



Note: You can create a composite index with many properties.

Retrieving indexes

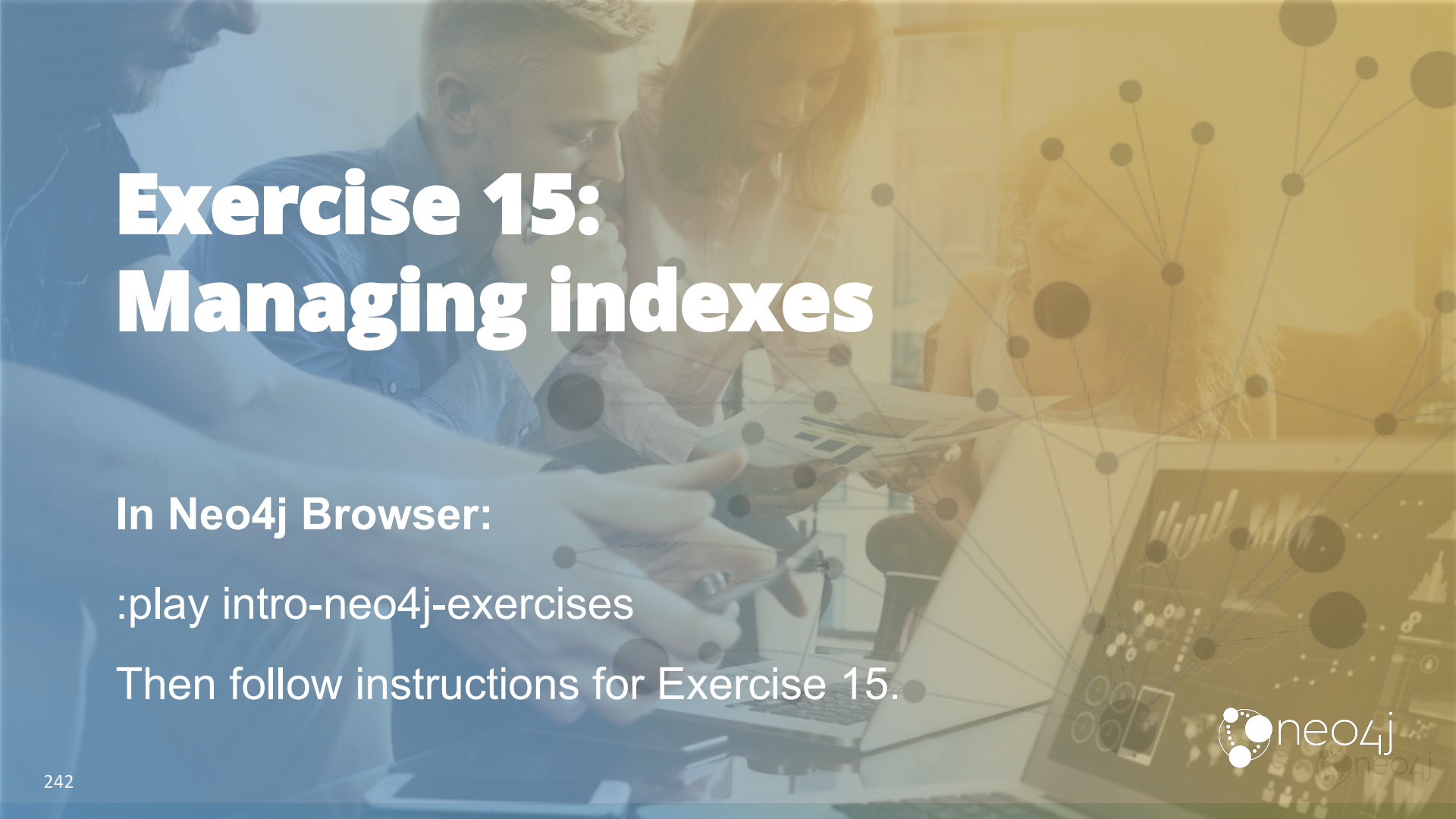
```
CALL db.indexes()
```



description	label	properties	state	type	provider
"INDEX ON :Movie(released)"	"Movie"	["released"]	"ONLINE"	"node_label_property"	{ "version": "2.0", "key": "lucene+native" }
"INDEX ON :Movie(released, videoFormat)"	"Movie"	["released", "videoFormat"]	"ONLINE"	"node_label_property"	{ "version": "2.0", "key": "lucene+native" }
"INDEX ON :Person(name, born)"	"Person"	["name", "born"]	"ONLINE"	"node_unique_property"	{ "version": "2.0", "key": "lucene+native" }
"INDEX ON :Movie(title)"	"Movie"	["title"]	"ONLINE"	"node_unique_property"	{ "version": "2.0", "key": "lucene+native" }

Dropping indexes

```
DROP INDEX ON :Movie(released, videoFormat)
```



Exercise 15: Managing indexes

In Neo4j Browser:

:play intro-neo4j-exercises

Then follow instructions for Exercise 15.

Importing data

CSV import is commonly used to import data into a graph where you can:

- Load data from a URL (http(s) or file).
- Process data as a stream of records.
- Create or update the graph with the data being loaded.
- Use transactions during the load.
- Transform and convert values from the load stream.
- Load up to 10M nodes and relationships.

Steps for importing data

1. Determine the number of lines that will be loaded.
   - Is the load possible without special processing to handle transactions?
2. Examine the data and see if it may need to be reformatted.
   - Does data need alterations based upon your data requirements?
3. Make sure reformatting you will do is correct.
   - Examine final formatting of data before loading it.
4. Load the data and create nodes in the graph.
5. Load the data and create the relationships in the graph.

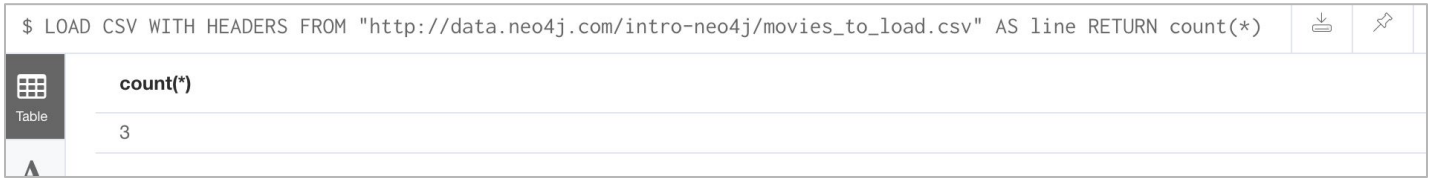
Importing normalized data - 1

Example CSV file, **movies_to_load.csv**:

```
id,title,country,year,summary
1,Wall Street,USA,1987, Every dream has a price.
2,The American President,USA,1995, Why can't the most powerful man in the world have the one thing he wants most?
3,The Shawshank Redemption,USA,1994, Fear can hold you prisoner. Hope can set you free.
```

1. Determine the number of lines that will be loaded:

```
LOAD CSV WITH HEADERS
FROM 'http://data.neo4j.com/intro-neo4j/movies_to_load.csv'
AS line
RETURN count(*)
```



Importing normalized data - 2

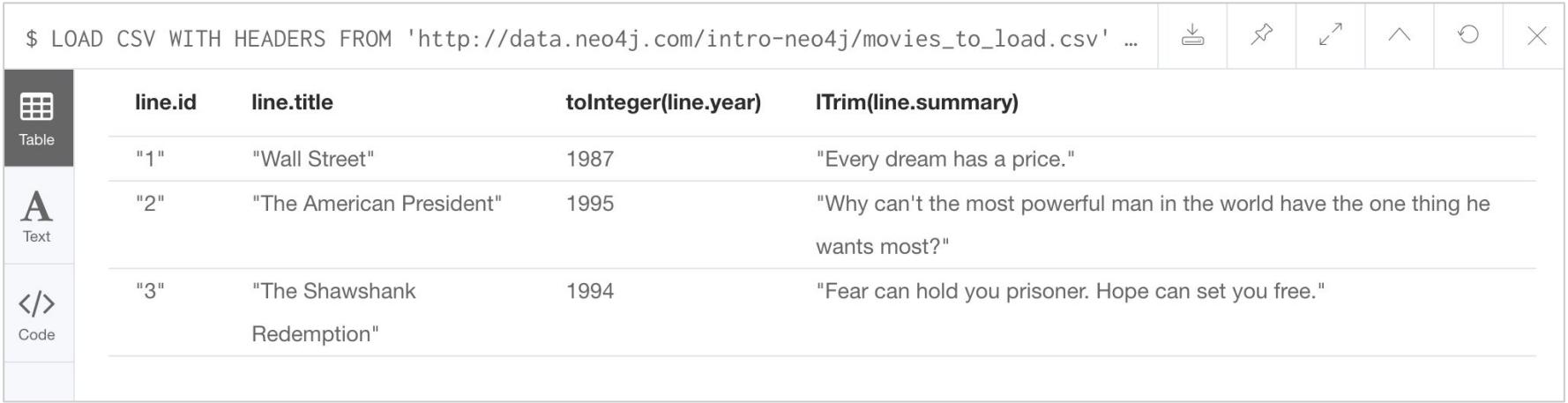
2. Examine the data and see if it may need to be reformatted:

```
LOAD CSV WITH HEADERS
FROM 'https://data.neo4j.com/intro-neo4j/movies_to_load.csv'
AS line
RETURN * LIMIT 1
```

Importing normalized data - 3

3. Format the data prior to loading:

```
LOAD CSV WITH HEADERS
FROM 'http://data.neo4j.com/intro-neo4j/movies_to_load.csv'
AS line
RETURN line.id, line.title, toInteger(line.year), trim(line.summary)
```



line.id	line.title	toInteger(line.year)	lTrim(line.summary)
"1"	"Wall Street"	1987	"Every dream has a price."
"2"	"The American President"	1995	"Why can't the most powerful man in the world have the one thing he wants most?"
"3"	"The Shawshank Redemption"	1994	"Fear can hold you prisoner. Hope can set you free."

Importing normalized data - 4

4. Load the data and create the nodes in the graph:

```
LOAD CSV WITH HEADERS
FROM 'https://data.neo4j.com/intro-neo4j/movies_to_load.csv'
AS line
CREATE (movie:Movie { movieId: line.id,
                      title: line.title,
                      released: toInteger(line.year) ,
                      tagline: trim(line.summary)})
```

```
$ LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/intro-neo4j/movies_to_load.csv" AS line CREATE (movie:Movie {…
```

Table

Added 3 labels, created 3 nodes, set 12 properties, completed after 289 ms.

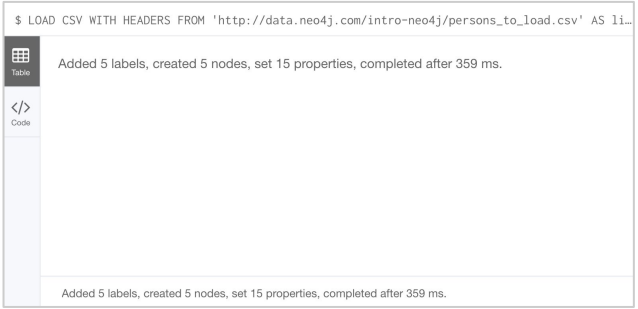
Importing the Person data

Example CSV file, **persons_to_load.csv**:

```
Id,name,birthyear
1,Charlie Sheen, 1965
2,Oliver Stone, 1946
3,Michael Douglas, 1944
4,Martin Sheen, 1940
5,Morgan Freeman, 1937
```

```
LOAD CSV WITH HEADERS
FROM 'https://data.neo4j.com/intro-neo4j/persons_to_load.csv'
AS line
MERGE (actor:Person { personId: line.Id })
ON CREATE SET actor.name = line.name,
              actor.born = toInteger(trim(line.birthyear))
```

We use MERGE to ensure that we will not create any duplicate nodes



Creating the relationships

Example CSV file, **roles_to_load.csv**:

```
personId,movieId,role
1,1,Bud Fox
4,1,Carl Fox
3,1,Gordon Gekko
4,2,A.J. MacInerney
3,2,President Andrew Shepherd
5,3,Ellis Boyd 'Red' Redding
```

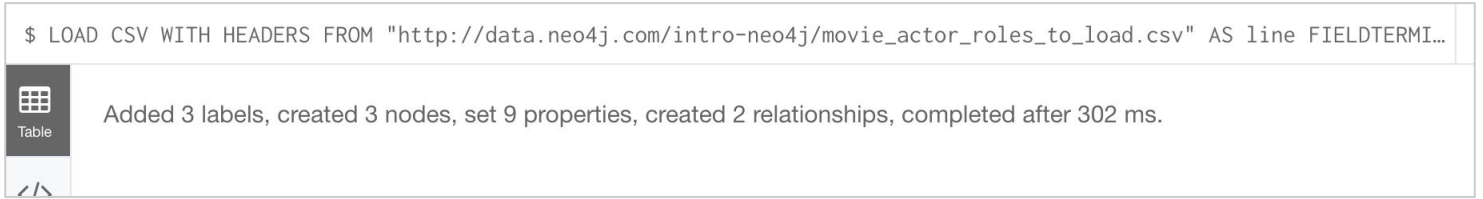
```
LOAD CSV WITH HEADERS
FROM 'https://data.neo4j.com/intro-neo4j/roles_to_load.csv'
AS line
MATCH (movie:Movie { movieId: line.movieId })
MATCH (person:Person { personId: line.personId })
CREATE (person)-[:ACTED_IN { roles: [line.role]}]->(movie)
```

Importing denormalized data

Example CSV file, **movie_actor_roles_to_load.csv**:

```
title;released;summary;actor;birthyear;characters
Back to the Future;1985;17 year old Marty McFly got home early last night. 30 years early.;Michael J. Fox;1961;Marty McFly
Back to the Future;1985;17 year old Marty McFly got home early last night. 30 years early.;Christopher Lloyd;1938;Dr. Emmet Brown
```

```
LOAD CSV WITH HEADERS
FROM 'https://data.neo4j.com/intro-neo4j/movie_actor_roles_to_load.csv'
AS line FIELDTERMINATOR ';'
MERGE (movie:Movie { title: line.title })
ON CREATE SET movie.released = toInteger(line.released),
              movie.tagline = line.summary
MERGE (actor:Person { name: line.actor })
ON CREATE SET actor.born = toInteger(line.birthyear)
MERGE (actor)-[r:ACTED_IN]->(movie)
ON CREATE SET r.roles = split(line.characters,',')
```



Importing a large dataset

< very large dataset that is greater than 10K rows>

```
PERIODIC COMMIT LOAD CSV . . .
```

Benefit: The graph engine will automatically commit data to avoid memory issues.

Exercise 16: Importing data

In Neo4j Browser:

:play intro-neo4j-exercises

Then follow instructions for Exercise 16.

Accessing Neo4j resources

There are many ways that you can learn more about Neo4j. A good starting point for learning about the resources available to you is the **Neo4j Learning Resources** page at https://neo4j.com/developer/resources/.


Check your understanding

Question 1

What Cypher keyword can you use to prefix any Cypher statement to examine how many db hits occurred when the statement executed?

Select the correct answer.

- ❑ ANALYZE
- ❑ EXPLAIN
- ❑ PROFILE
- ❑ MONITOR

Answer 1

What Cypher keyword can you use to prefix any Cypher statement to examine how many db hits occurred when the statement executed?

Select the correct answer.

- ❏ ANALYZE
- ❏ EXPLAIN
- ✓ PROFILE
- ❏ MONITOR

Question 2

What types of constraints can you define for a graph that are asserted when a node or relationship is created or updated?

Select the correct answers.

- ❏ unique values for a property of a node
- ❏ unique values for a property of a relationship
- ❏ a node must have a certain set of properties with values
- ❏ a relationship must have a certain set of properties with values

Answer 2

What types of constraints can you define for a graph that are asserted when a node or relationship is created or updated?

Select the correct answers.

- ✓ unique values for a property of a node
- ❑ unique values for a property of a relationship
- ✓ a node must have a certain set of properties with values
- ✓ a relationship must have a certain set of properties with values

Question 3

In general, what is the maximum number of nodes or relationships that you can easily create using LOAD CSV?

Select the correct answer.

- ❑ 1K
- ❑ 10K
- ❑ 1M
- ❑ 10M

Answer 3

In general, what is the maximum number of nodes or relationships that you can easily create using LOAD CSV?

Select the correct answer.

- ❑ 1K
- ❑ 10K
- ❑ 1M
- ✓ 10M

Summary

You should be able to:

- Use parameters in your Cypher statements.
- Analyze Cypher execution.
- Monitor queries.
- Manage constraints and node keys for the graph.
- Import data into a graph from CSV files.
- Manage indexes for the graph.
- Access Neo4j resources.

Course feedback

We value your feedback!

Please fill out the feedback form and receive a

Certificate of Completion

bit.ly/neo-survey