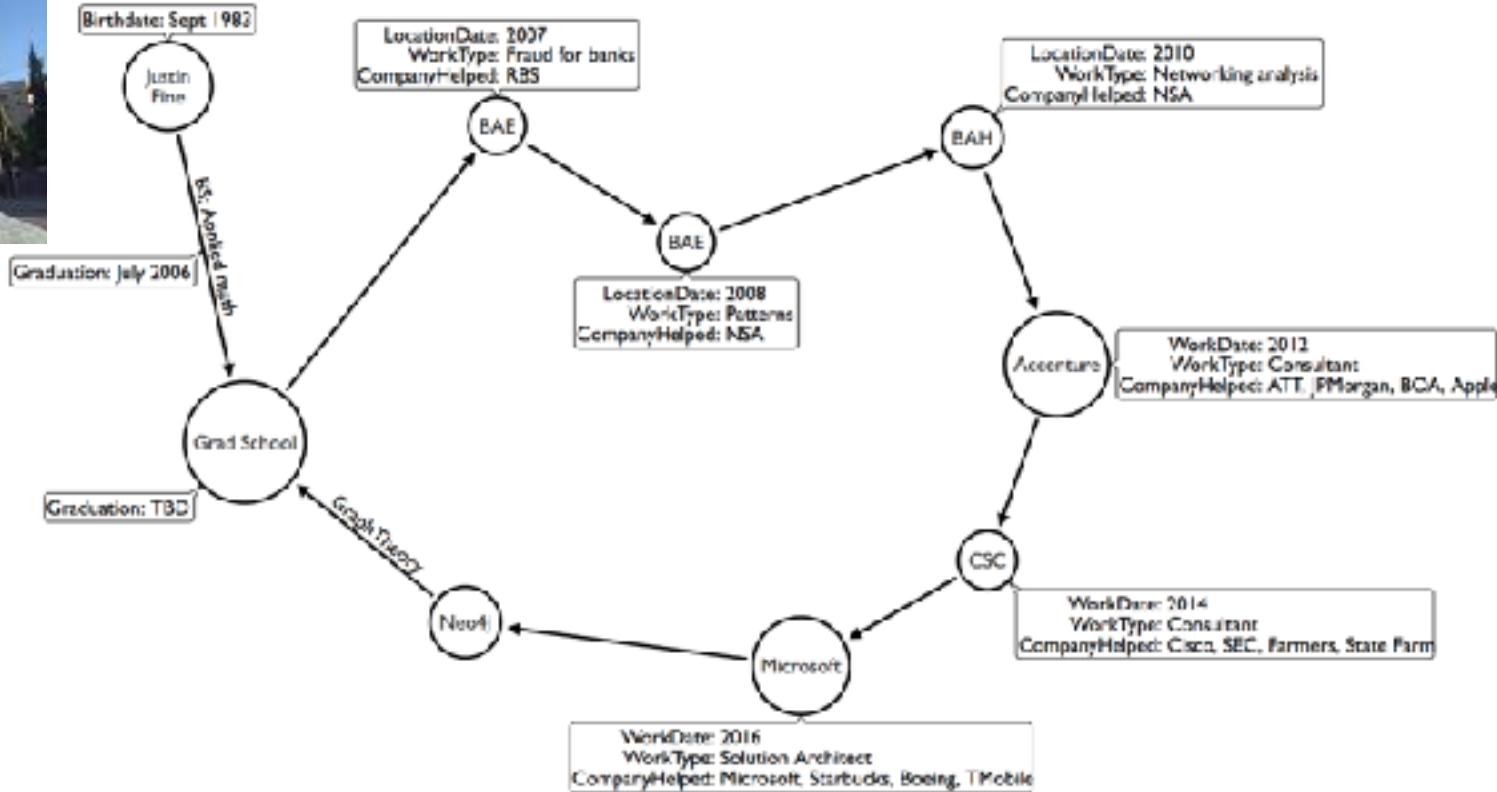


A photograph of a person's hands holding and fitting together yellow puzzle pieces. Overlaid on the image is a network graph consisting of numerous dark grey circular nodes connected by thin grey lines, representing relationships or data points.

# Introduction to Graph Databases

v 1.0



# 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.

- Node (vertices)
- Relationship (edge)
- Degree
- Graph (G)
- Hops
- Weighted Graph
- Directed Graph

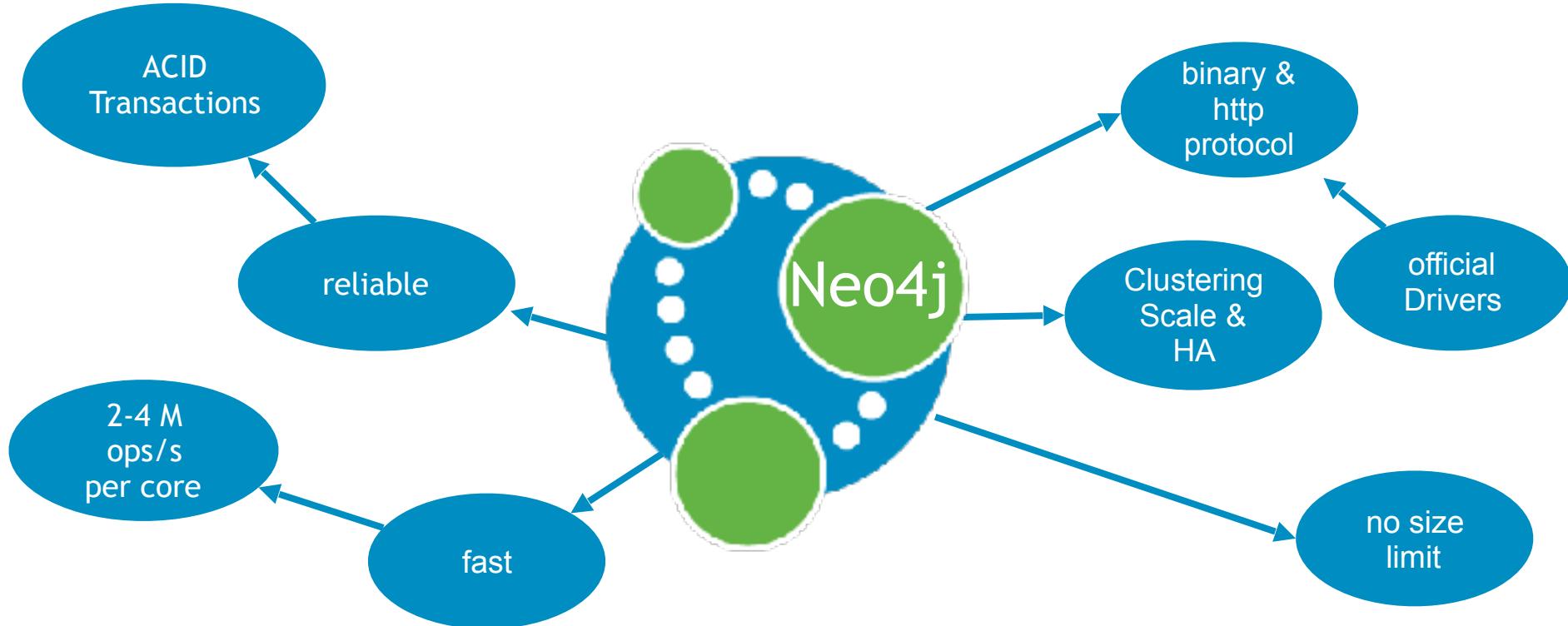
# Cool Graph Background



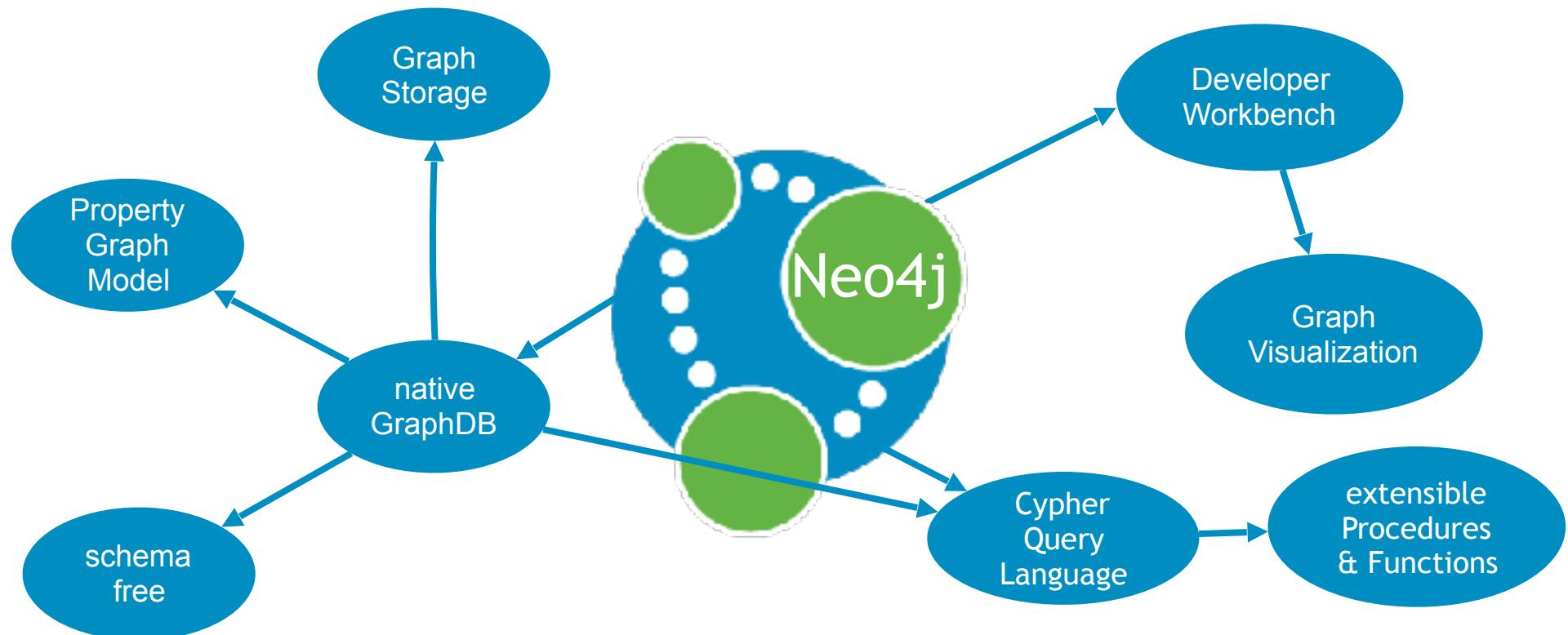
- 7 Bridge problem
- Erdos Number (Erdos-Bacon Number)
- Feynman Graph (QED)



# Neo4j is a database



# Neo4j is a graph database



# 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

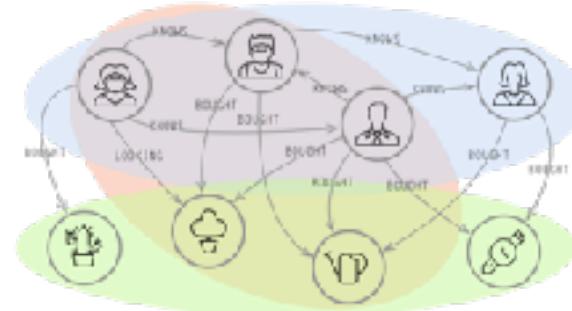
## Internal Applications

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



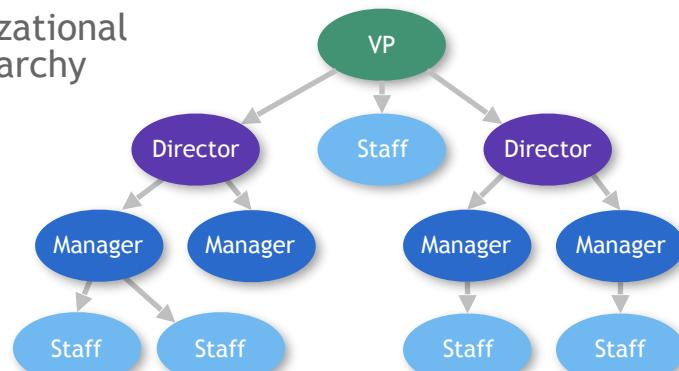
## Customer-Facing Applications

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



# Use cases: Master data management.

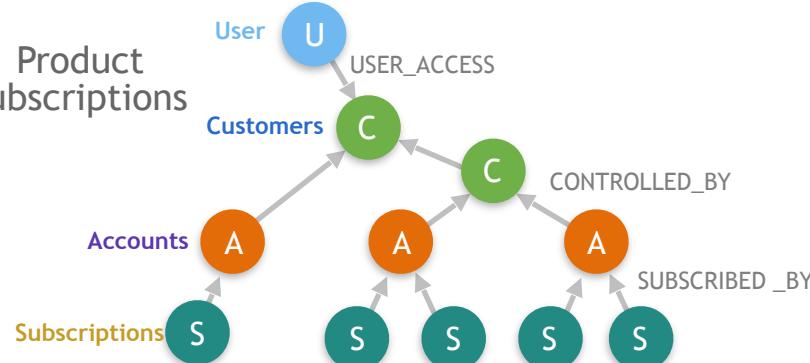
Organizational Hierarchy



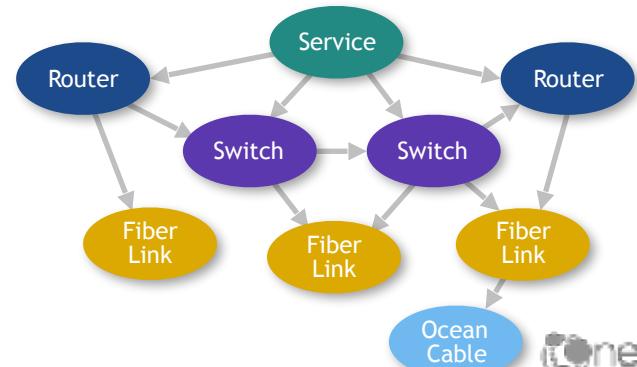
Customer 360



Product Subscriptions



CMDB Network Inventory

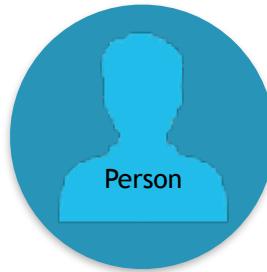


# What is a graph?

- Nodes
- Relationships
- Properties
- Labels

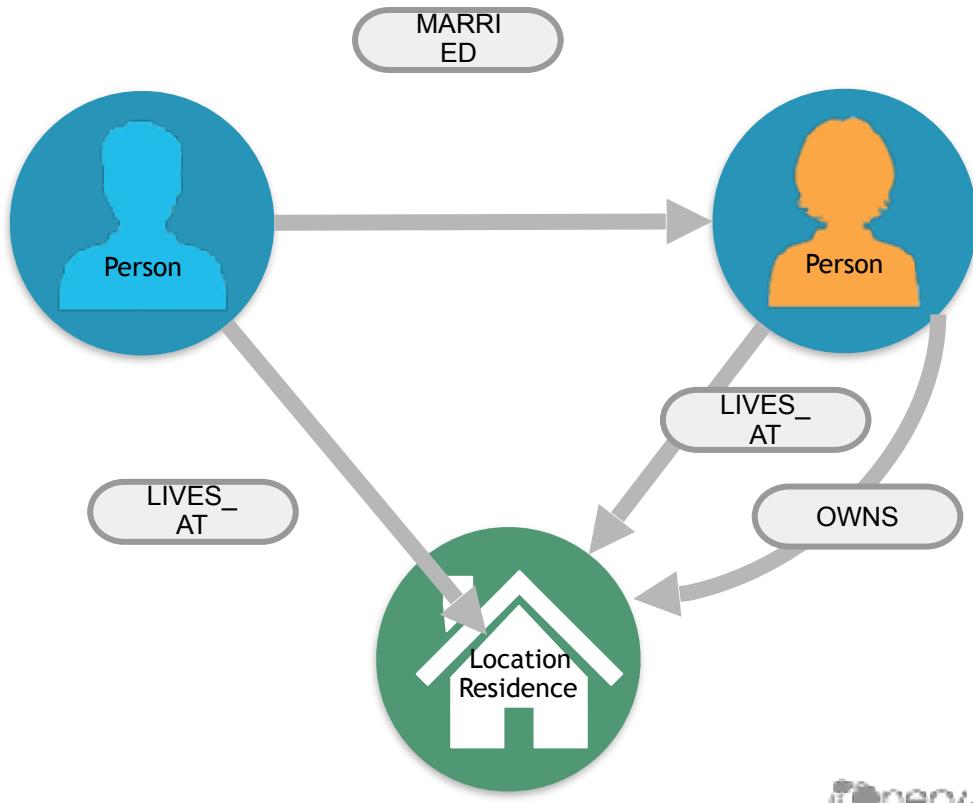
# Nodes

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



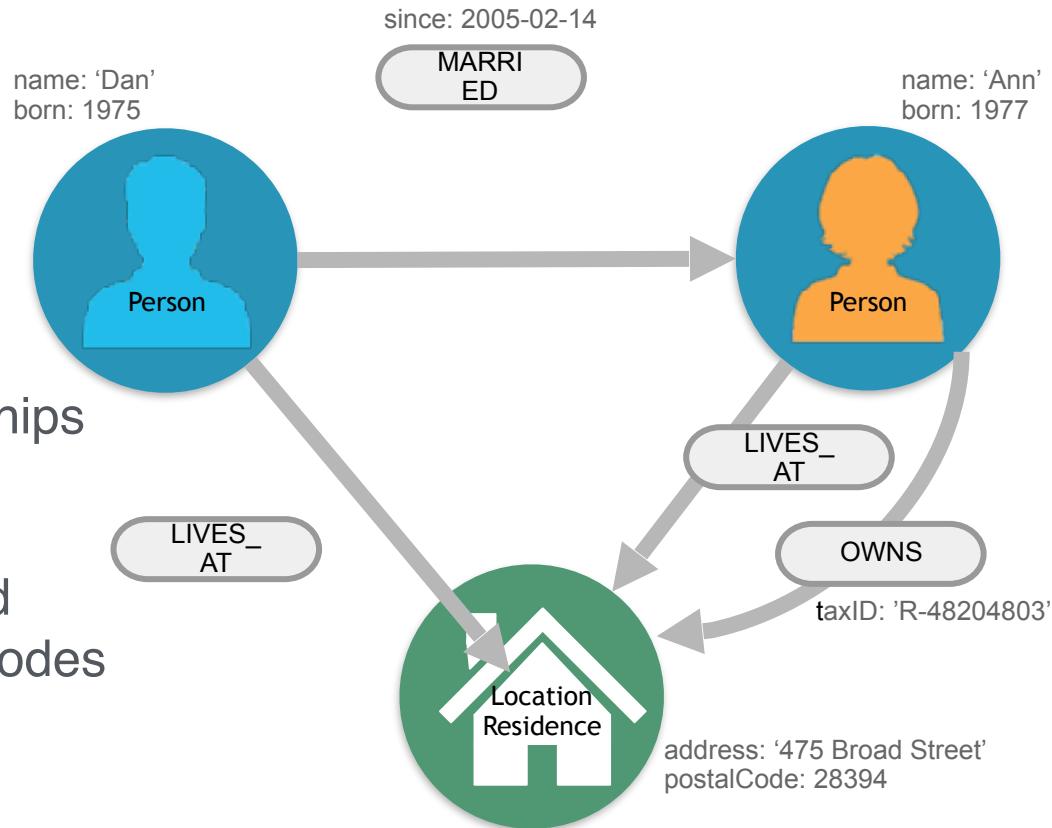
# Relationships

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

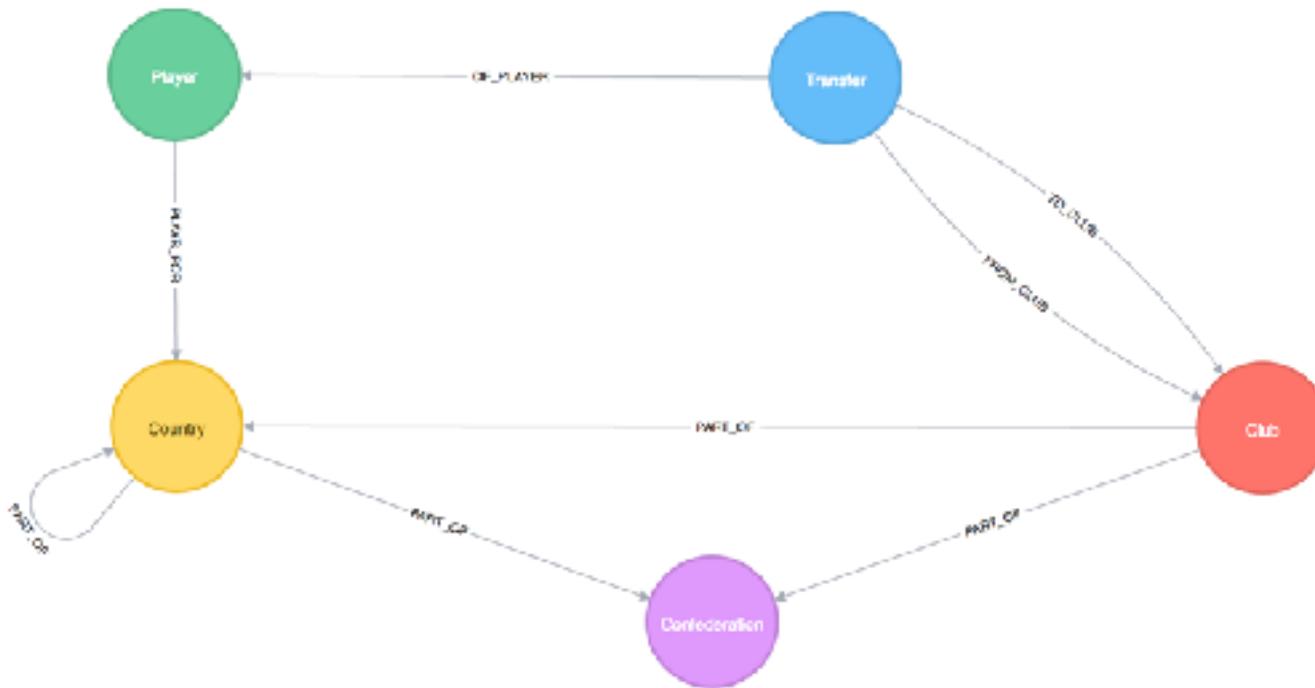


# Properties

- 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 data model: CALL db.schema()





# Introduction to Neo4j

v 1.0



# Overview

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

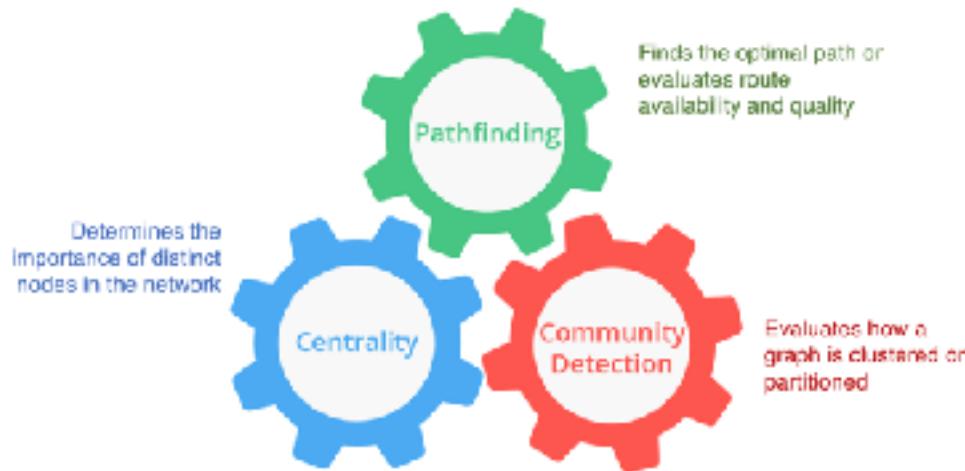
- Describe the components and benefits of the Neo4j Graph Platform.

# Libraries

Out-of-the-box:

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

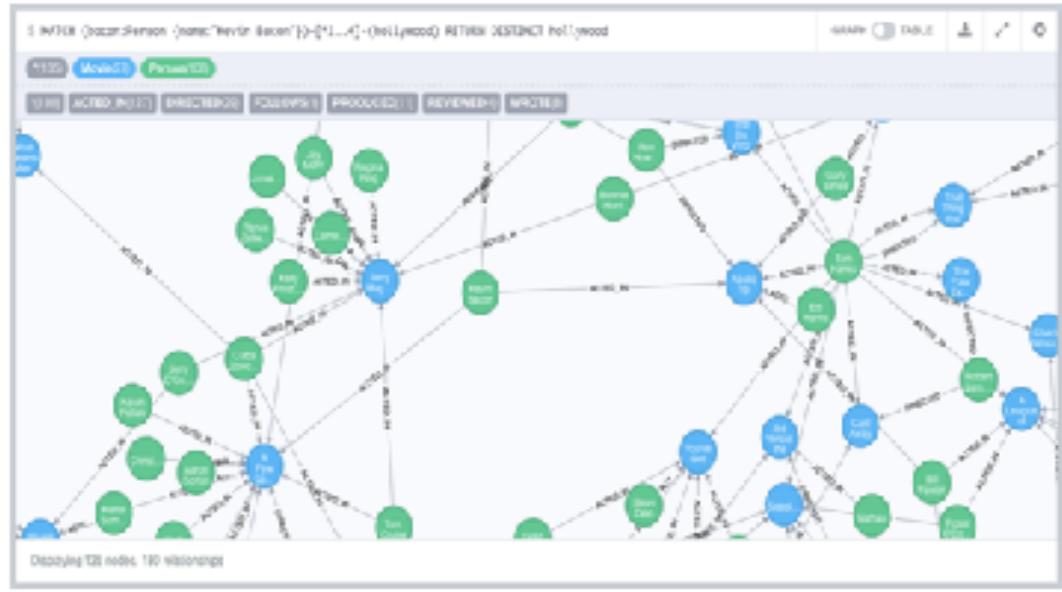
Neo4j community has contributed many specialized libraries also.



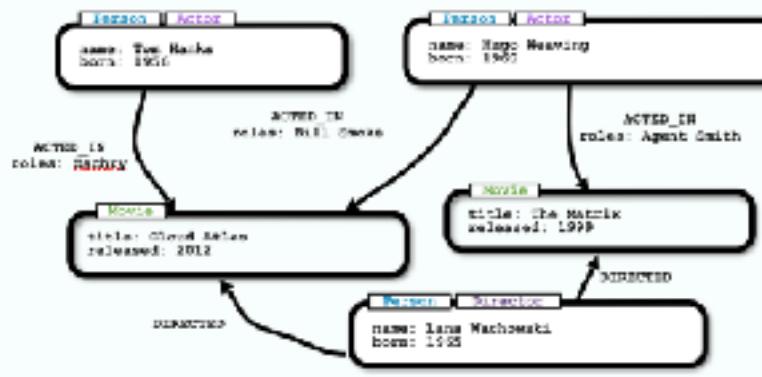
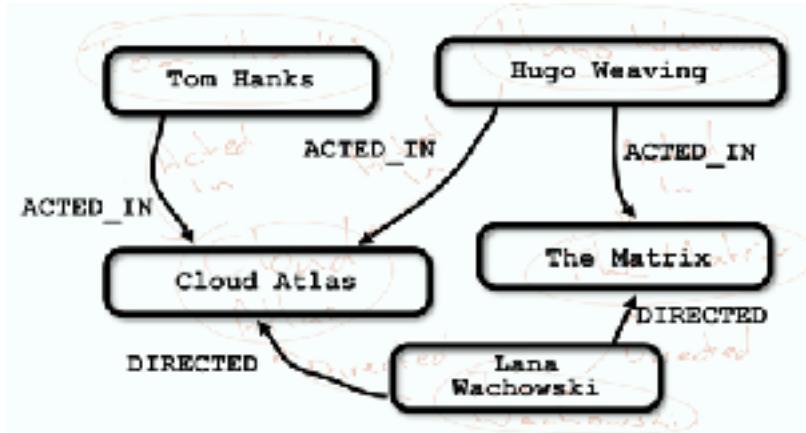
# Tools

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

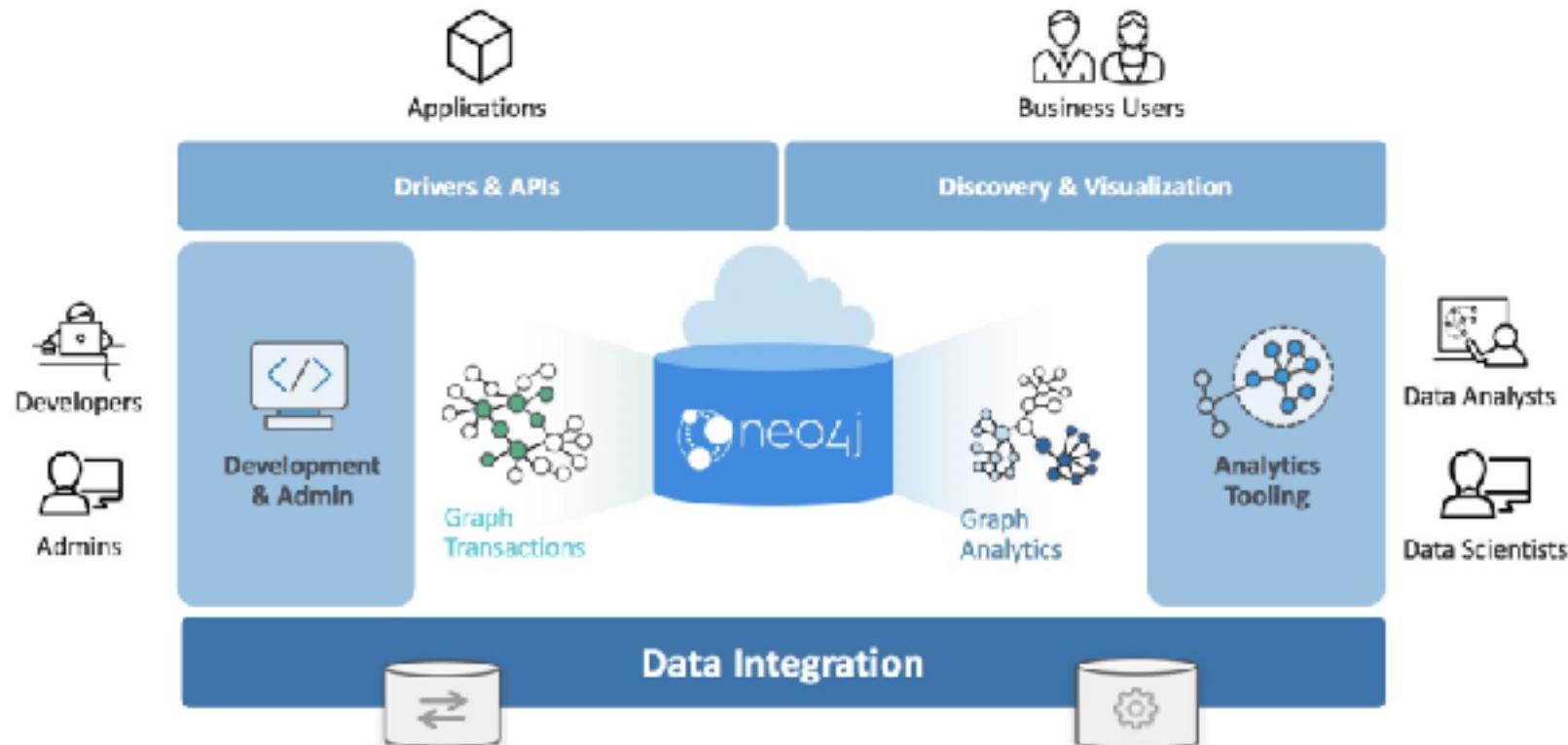
Neo4j community has contributed many specialized tools also.



# Whiteboard modeling



# Neo4j Graph Platform architecture



# Introduction to Cypher

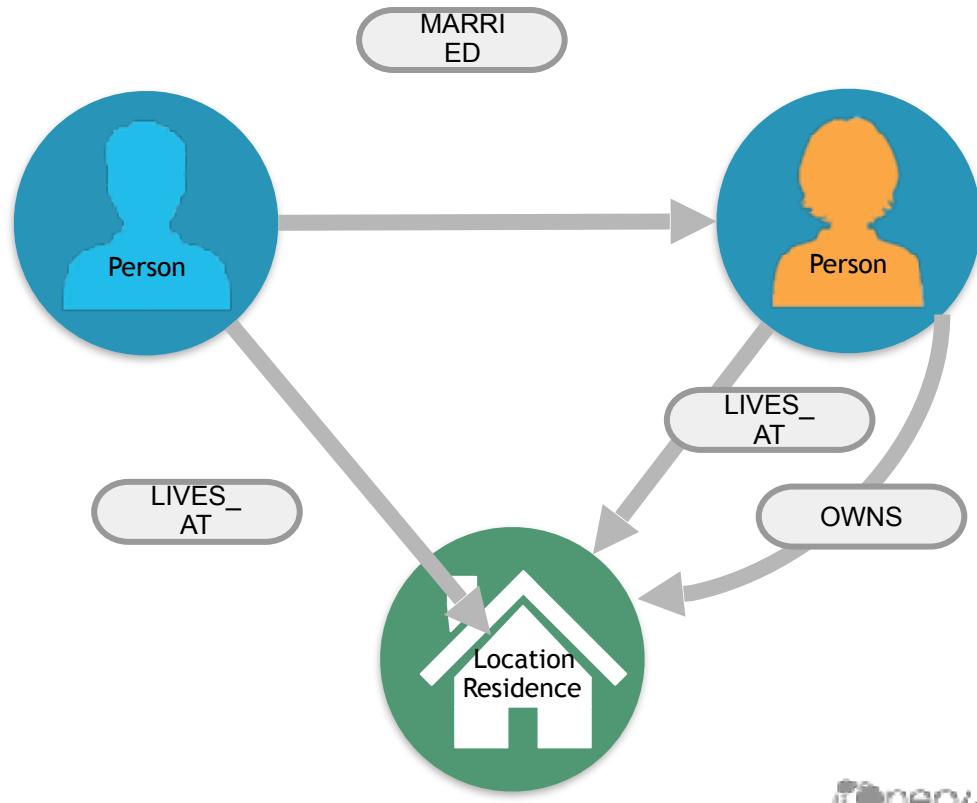
# Overview

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

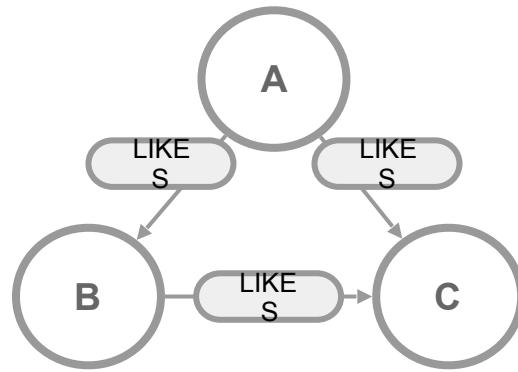
- Retrieve nodes from the graph.
- Filter nodes retrieved using labels and property values of nodes.
- Retrieve property values from nodes in the graph.
- Filter nodes retrieved using relationships.

# What is Cypher?

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



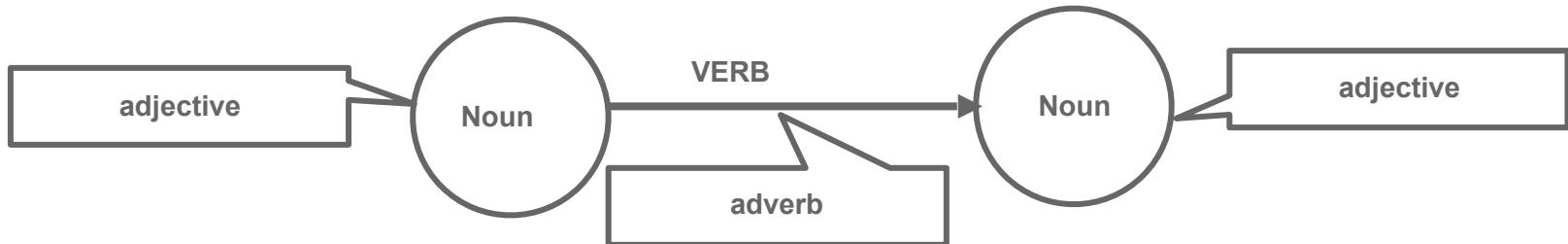
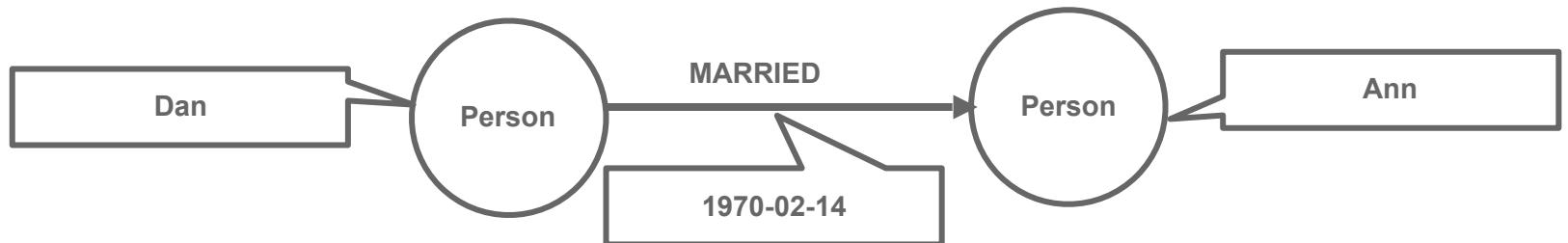
# Cypher is ASCII Art



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

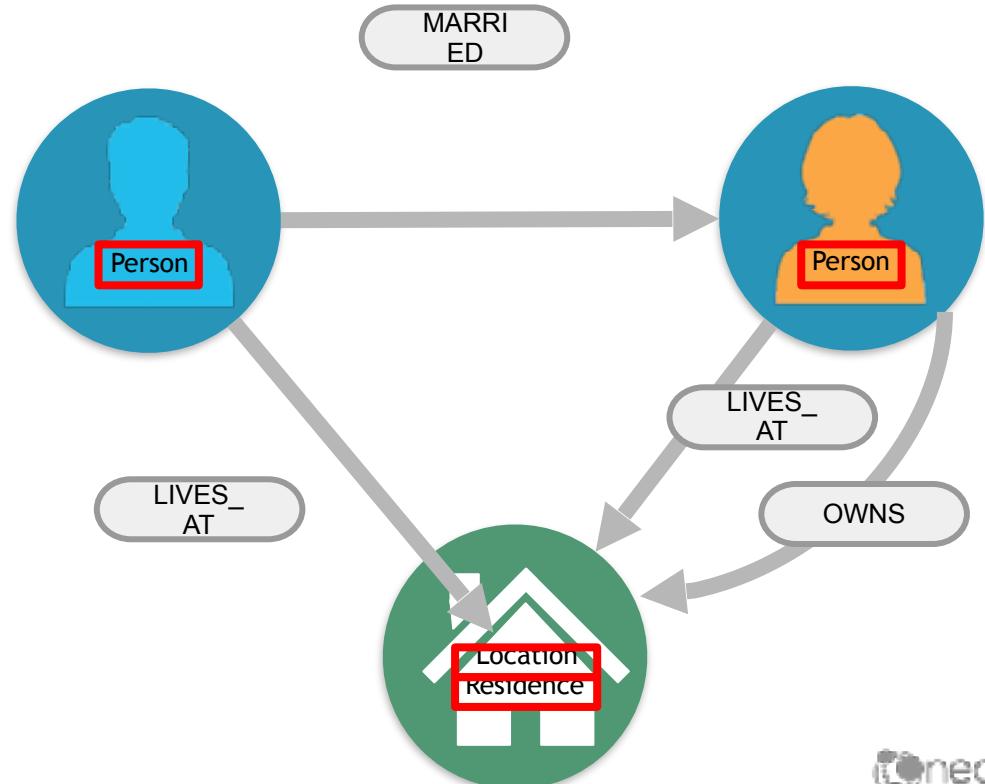
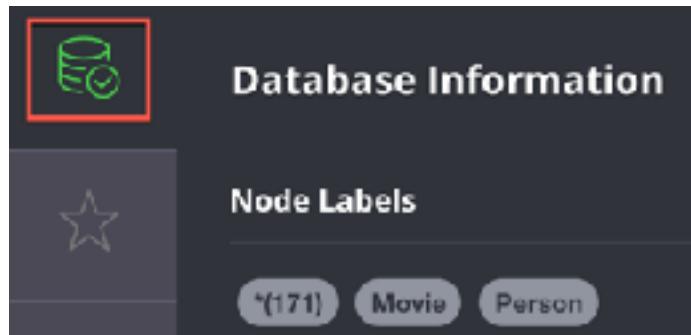
```
(A) - [ :LIKES ] -> (B) - [ :LIKES ] -> (C) <- [ :LIKES ] - (A)
```

# Cypher is readable



# Labels

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



# Comments in Cypher

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

# Using MATCH to retrieve nodes

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

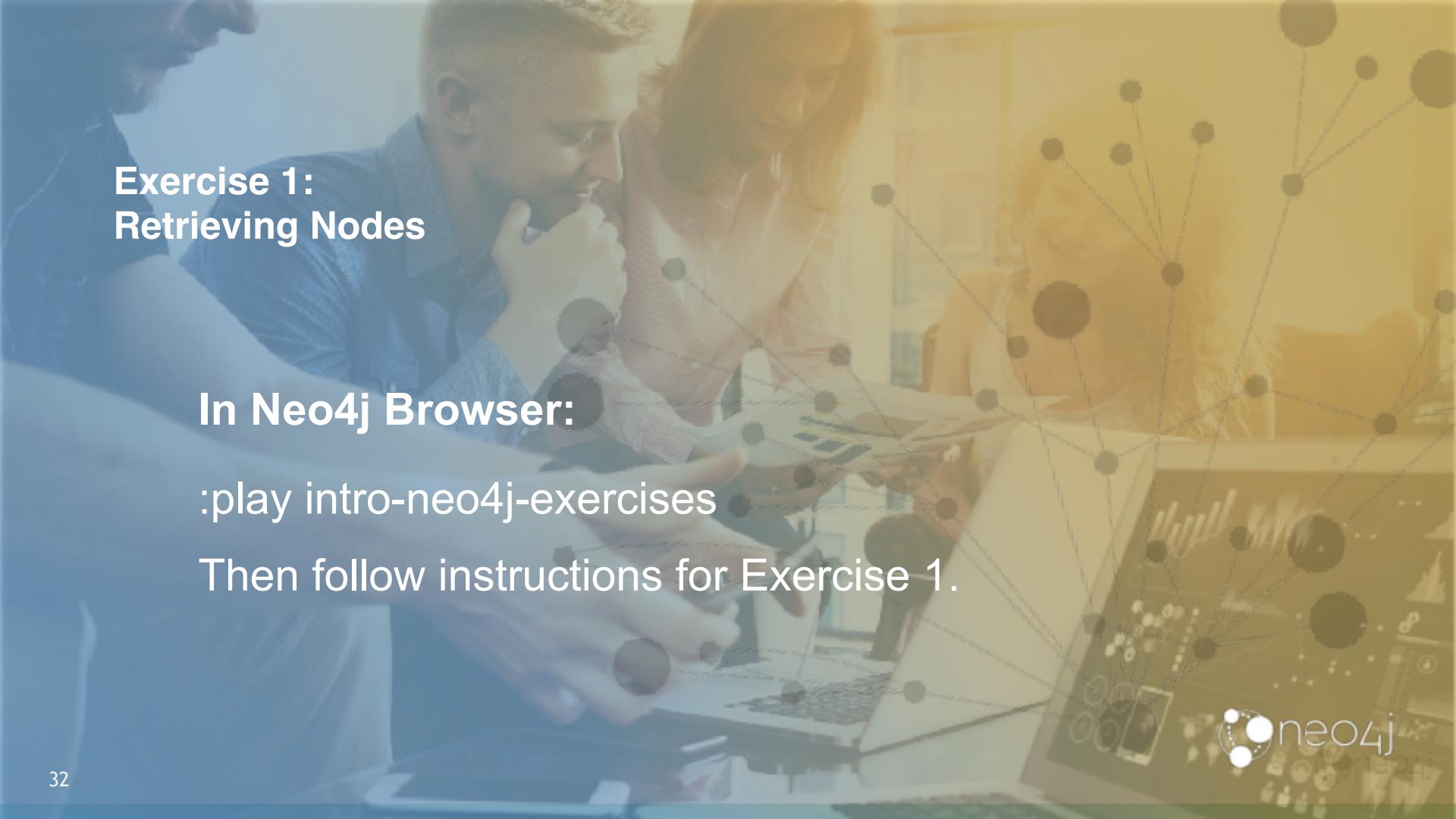
```
MATCH (p:Person) // returns all Person nodes in the graph  
RETURN p
```



# Viewing nodes as table data

The screenshot shows the Neo4j Browser interface with the following details:

- Query Bar:** Displays the query: `$ MATCH (p:Person) RETURN p`.
- Toolbar:** Includes icons for back, forward, search, and close.
- Left Sidebar:** Features a "Graph" icon (highlighted with a red box), a "Table" icon (highlighted with a red box), and an "A" icon labeled "Text". Below these are "Code" and "Code" buttons.
- Result Area:** Shows three nodes represented as tables:
  - Node 1:** Contains the JSON object: `{ "name": "Keanu Reeves", "born": 1964 }`.
  - Node 2:** Contains the JSON object: `{ "name": "Carrie-Anne Moss", "born": 1969 }`.
  - Node 3:** Contains the JSON object: `{ "name": "Laurence Fishburne", "born": 1961 }`.

A photograph of two people, a man and a woman, sitting at a desk in an office environment. They are looking at a computer screen together, possibly discussing data. A network graph with nodes and connections is overlaid on the right side of the image.

## Exercise 1: Retrieving Nodes

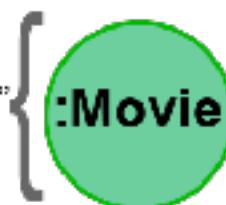
In Neo4j Browser:

:play intro-neo4j-exercises

Then follow instructions for Exercise 1.

# Properties

**title:** "Something's Gotta Give"  
**released:** 2003



**title:** 'V for Vendetta'  
**released:** 2006  
**tagline:** 'Freedom! Forever!'



**title:** 'The Matrix Reloaded'  
**released:** 2003  
**tagline:** 'Free your mind'

# Retrieving nodes filtered by a property value - 1

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

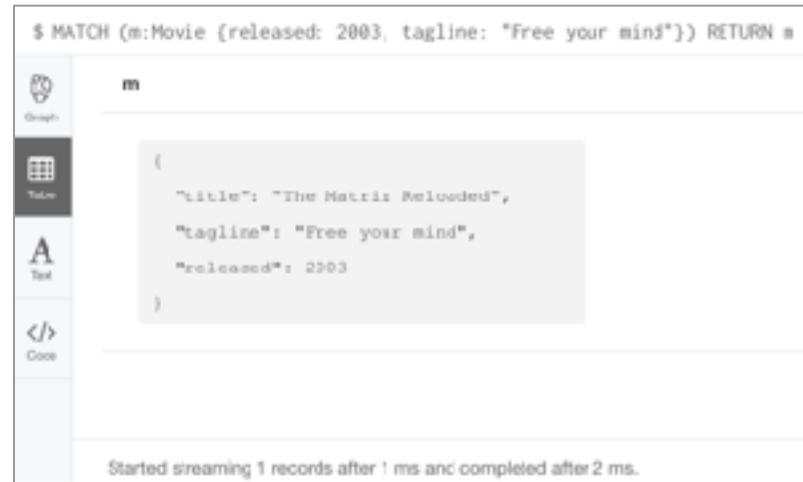
```
MATCH (p:Person {born: 1970})  
RETURN p
```



# 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
```



The screenshot shows the Neo4j browser interface with the following details:

- Query:** \$ MATCH (m:Movie {released: 2003, tagline: "Free your mind"}) RETURN m
- Graph View:** Shows a small graph visualization of the movie node.
- Table View:** Shows the results in a table format:

m
{ "title": "The Matrix Reloaded", "tagline": "Free your mind", "released": 2003 }

- Text View:** Shows the JSON representation of the node:

```
{"title": "The Matrix Reloaded",  
 "tagline": "Free your mind",  
 "released": 2003}
```
- Code View:** Shows the executed Cypher query.
- Status Bar:** Started streaming 1 records after 1 ms and completed after 2 ms.

# Returning property values

Find all people born in 1965 and return their names:

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



The screenshot shows the Neo4j browser interface with a query results table. On the left, there are three vertical tabs: 'Tables' (selected), 'Nodes' (with a count of 1), and 'Relationships' (with a count of 1). The main area displays a table with two columns: 'p.name' and 'p.born'. The table contains three rows of data:

p.name	p.born
"Lana Wachowski"	1965
"Tom Tykwer"	1965
"John C. Reilly"	1965

At the bottom of the table, a status message reads: "Started streaming 3 records in less than 1 ms and completed after 1 ms."

# Specifying aliases

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

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



Table

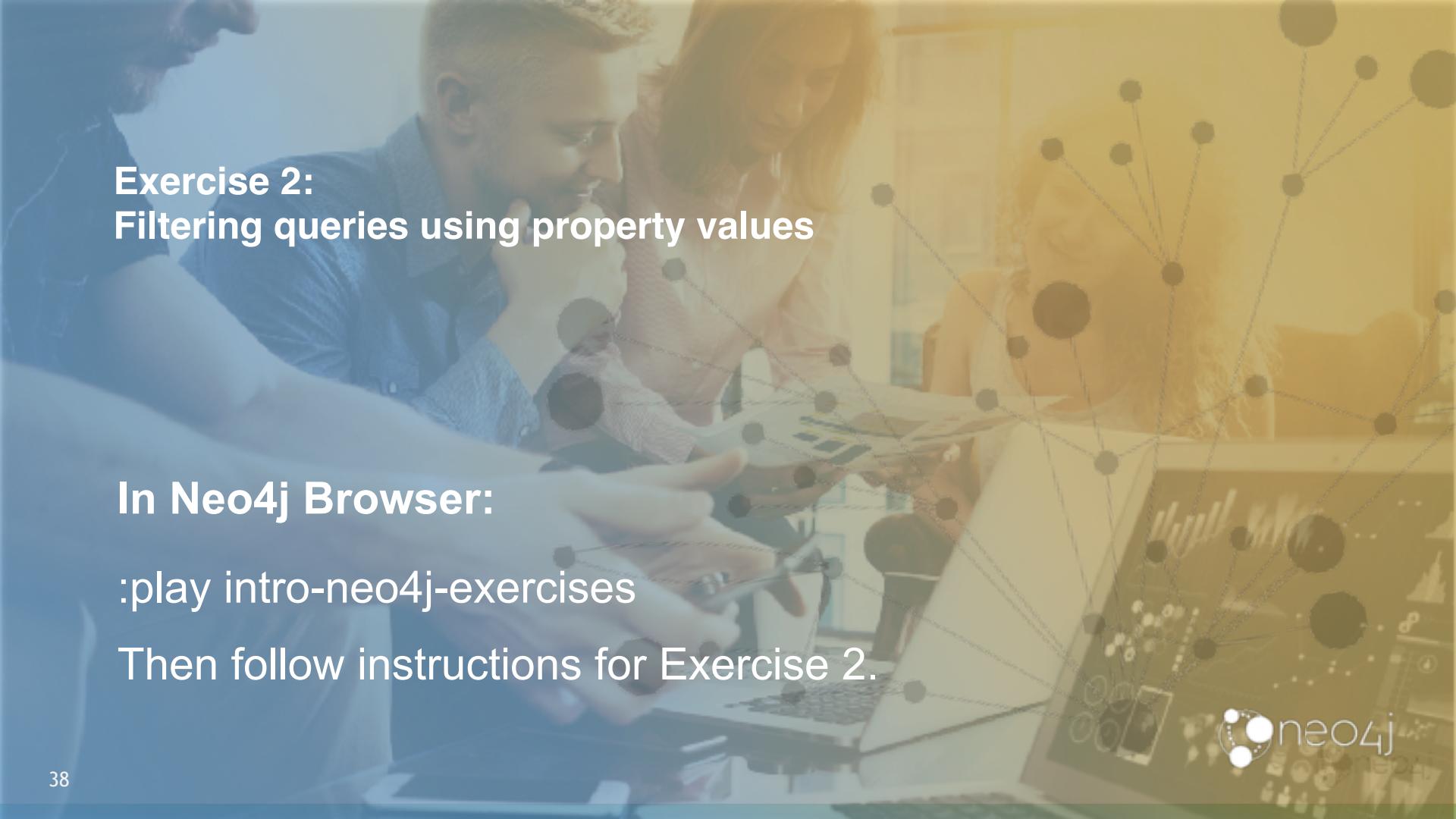


Text



Code

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

A photograph of two people, a man and a woman, sitting at a desk in an office environment. They are looking down at a laptop screen, possibly discussing something. A large, semi-transparent network graph with nodes and connections is overlaid on the right side of the image.

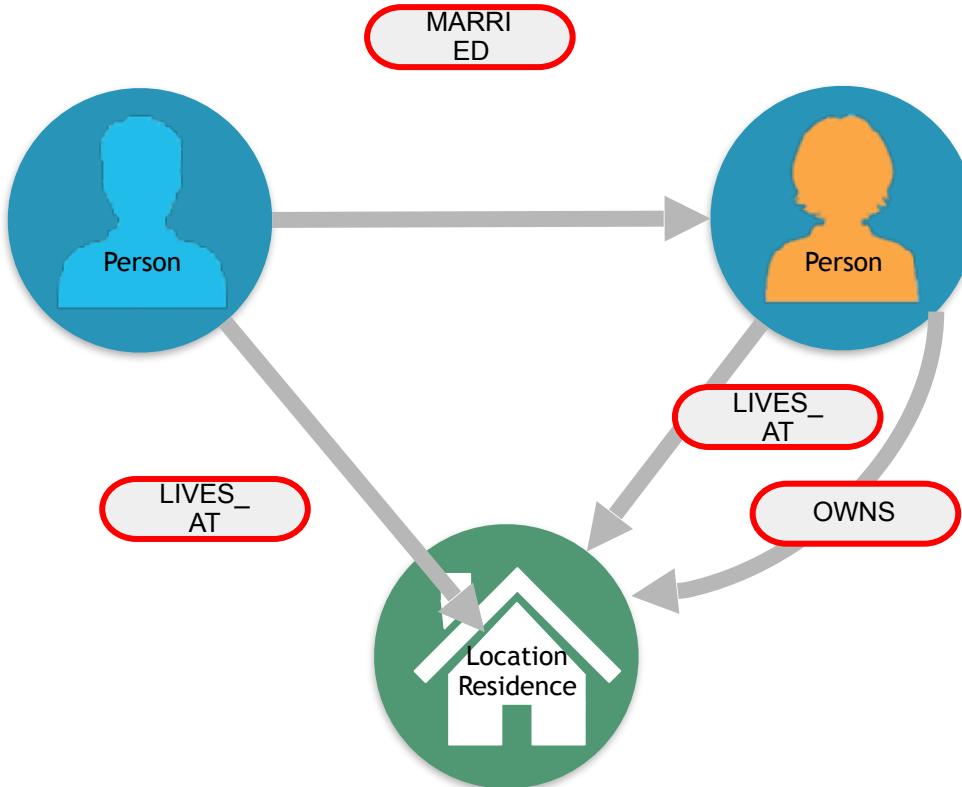
## **Exercise 2:** **Filtering queries using property values**

**In Neo4j Browser:**

```
:play intro-neo4j-exercises
```

Then follow instructions for Exercise 2.

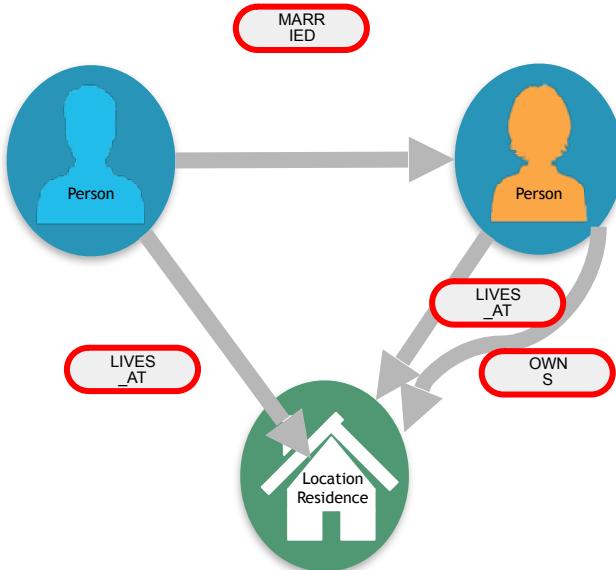
# Relationships



# 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
```

# Querying by multiple relationships

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

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

\$ MATCH (p:Person {name: 'Tom Hanks'})-[:ACTED_IN   :DIRECTED]->(m:Movie) RETURN p.name, m.title	
p.name	m.title
"Tom Hanks"	"Apollo 13"
"Tom Hanks"	"Cast Away"
"Tom Hanks"	"The Polar Express"
"Tom Hanks"	"A League of Their Own"
"Tom Hanks"	"Charlie Wilson's War"
"Tom Hanks"	"Ghosts Adams"
"Tom Hanks"	"The Da Vinci Code"
"Tom Hanks"	"The Green Mile"
"Tom Hanks"	"You've Got Mail"
"Tom Hanks"	"That Thing You Do"
"Tom Hanks"	"That Thing You Do"
"Tom Hanks"	"Joe Versus the Volcano"
"Tom Hanks"	"Sleepless in Seattle"

Started streaming 10 records after 1 ms and completed after 1 ms.

# Using an anonymous relationship for a query

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

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



Connect result  
nodes enabled in  
Neo4j Browser

# Retrieving relationship types

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

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

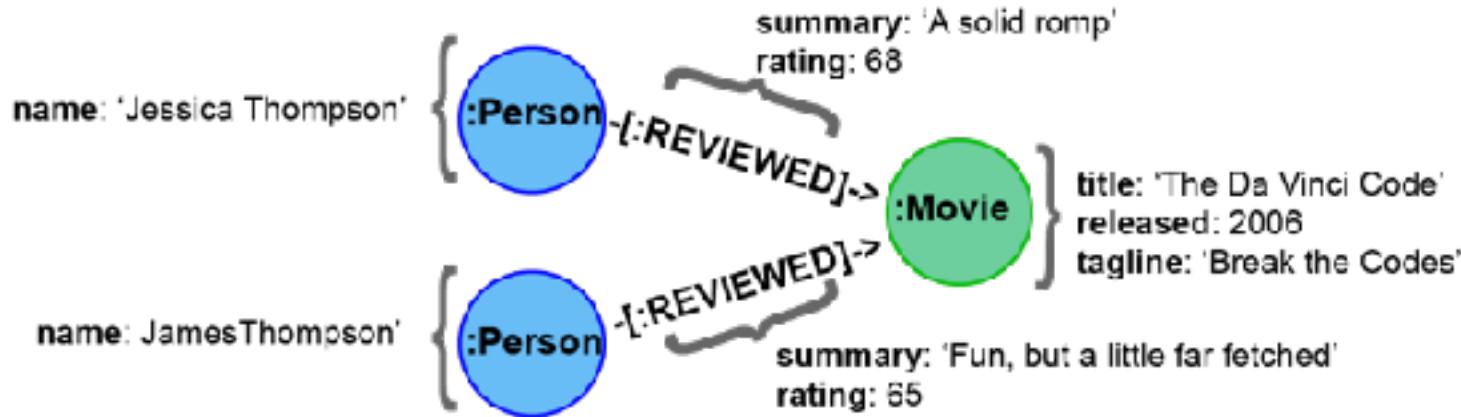
\$ MATCH (p:Person)-[rel]->(:Movie {title:'The Matrix'}) RETURN p.name, type(rel)

The screenshot shows the Neo4j browser interface with a query results table. The table has two columns: 'p.name' and 'type(rel)'. The data rows are as follows:

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

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

# Retrieving properties for a relationship - 1



## Retrieving properties for a relationship - 2

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

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

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



Table

p.name

"James Thompson"



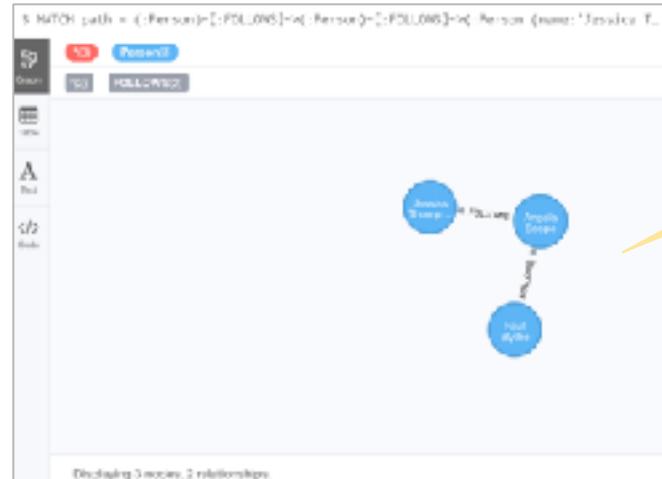
Text

# 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
```



# Getting More Out of Queries

# 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
```

name	title	released
Halle Berry	'The Matrix Reloaded'	2003
Laurence Fishburne	'The Matrix Revolutions'	2003
Keanu Reeves	'The Matrix Reloaded'	2003
Halle Berry	'The Matrix Revolutions'	2003
Laurence Fishburne	'The Matrix Revolutions'	2003
Keanu Reeves	'The Matrix Revolutions'	2003
Halle Berry	'The Matrix Reloaded'	2002
Laurence Fishburne	'The Matrix Revolutions'	2002
Keanu Reeves	'The Matrix Revolutions'	2002
Jack Palance	'The Polar Express'	2004
Dana Mastro	'Something's Gotta Give'	2003
Keanu Reeves	'Something's Gotta Give'	2003
Tom Hanks	'The Polar Express'	2004

Started streaming 12 records after 1 ms and completed after 8 ms.

# 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
```

Nodes

Relationships

Labels

Properties

Script

Code

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."
"Hefta"	"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 p.name STARTS WITH 'Michael' RETURN p.name
```

Table
A
</>

p.name

'Michael Clarke Duncan'

'Michael Sheen'

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

## 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
```

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



p.name

"Aaron Sorkin"

m.title

"A Few Good Men"



"Jim Cash"

"Top Gun"



"David Mitchell"

"Cloud Atlas"

# Testing with list values - 1

Find all people born in 1965 or 1970:

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

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



name	yearBorn
"Lara Wachowski"	1965
"Jay Mohr"	1970
"River Phoenix"	1970
"Ethan Hawke"	1970
"Brooke Langton"	1970
"Tom Tykwer"	1965
"John G. Reilly"	1965

Started streaming 7 records after 1 ms and completed after 2 ms.

## 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
```

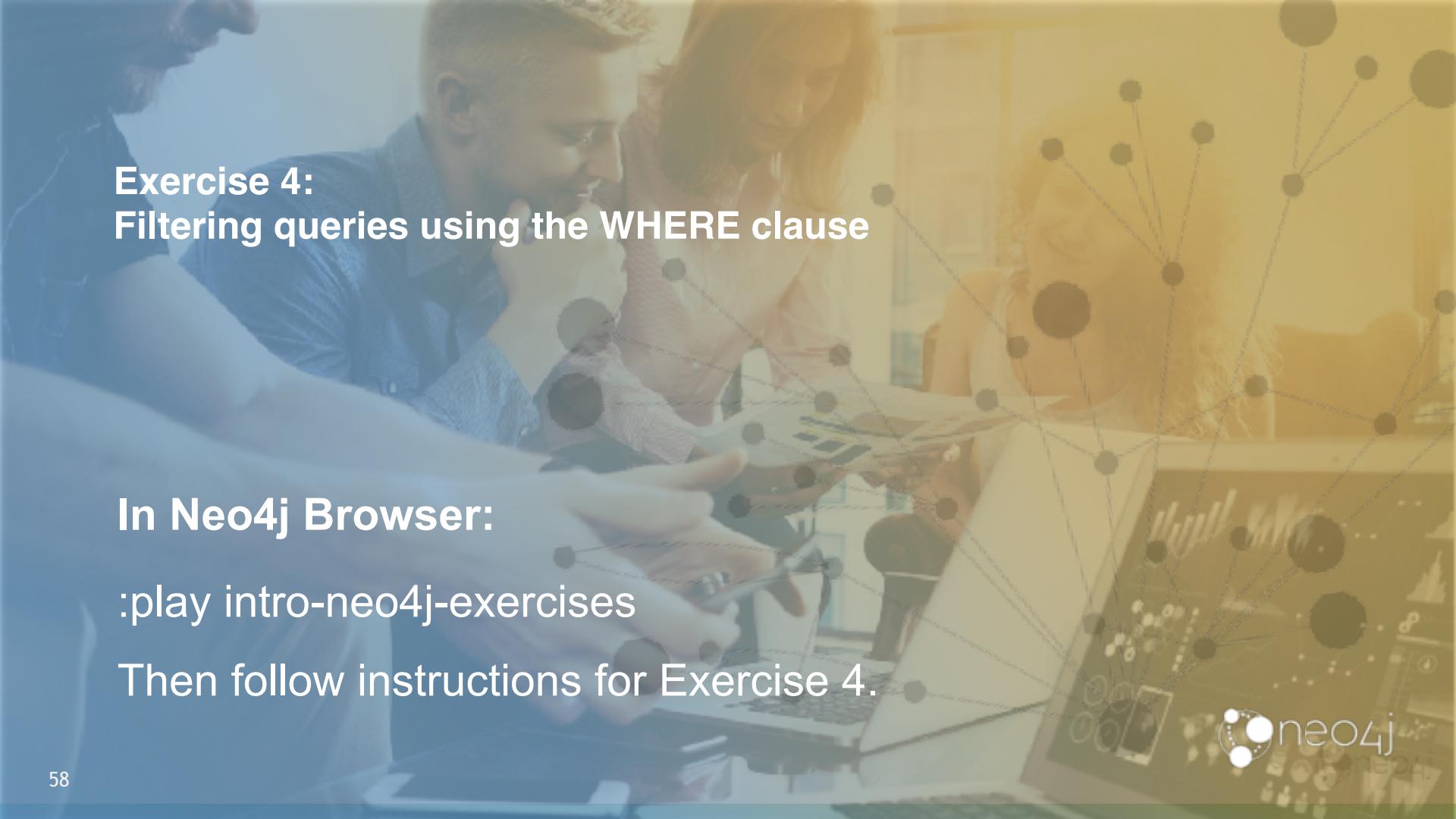


Table

p.name

"Keanu Reeves"

A

A background photograph of three people working at a desk in an office setting. A network graph with nodes and connections is overlaid on the right side of the image.

## Exercise 4: Filtering queries using the WHERE clause

In Neo4j Browser:

:play intro-neo4j-exercises

Then follow instructions for Exercise 4.

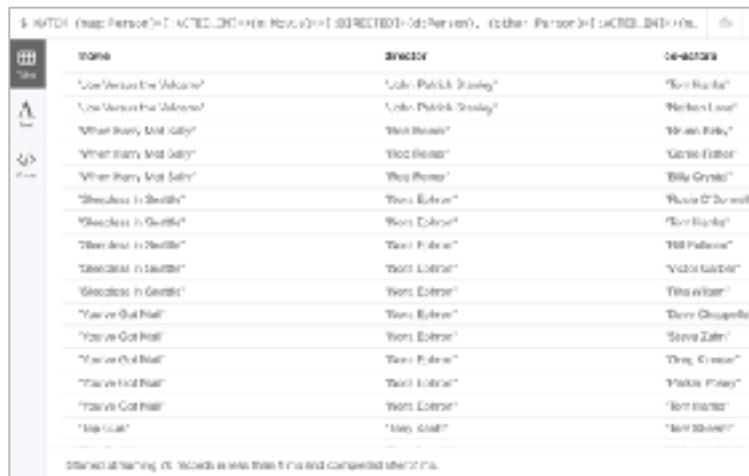
# Controlling query processing

- 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

# Example : 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`
```

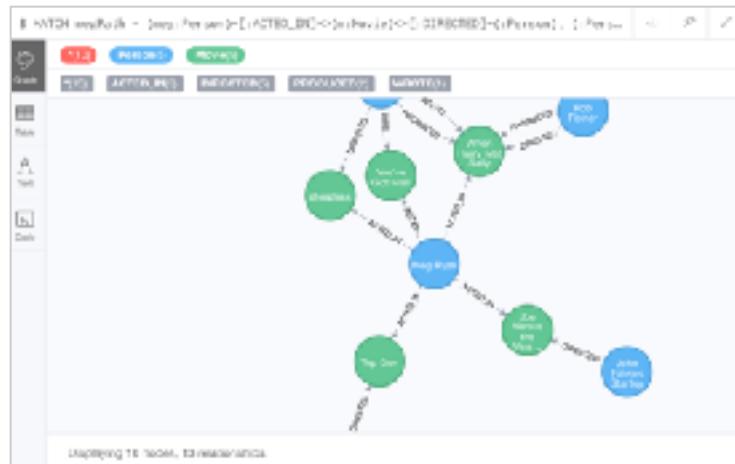


movie	director	co-actors
"You Haven't Seen the Last One"	John Patrick Stanley	"Keanu Reeves"
"You Haven't Seen the Last One!"	John Patrick Stanley	"Keanu Reeves"
"When Harry Met Sally"	Rob Reiner	"Megan Mullally"
"When Harry Met Sally"	Rob Reiner	"Gwyneth Paltrow"
"When Harry Met Sally"	Rob Reiner	"Billy Crystal"
"Shakespeare in Love"	Tom Cruise	"Hilary Duff"
"Shakespeare in Love"	Tom Cruise	"Keanu Reeves"
"Bridget Jones's Diary"	Mike Newell	"Tina Fey"
"Bridget Jones's Diary"	Mike Newell	"Viggo Mortensen"
"Bridget Jones's Diary"	Mike Newell	"Naomi Campbell"
"Bridget Jones's Diary"	Mike Newell	"Tilda Swinton"
"You've Got Mail"	Mike Newell	"Demi Moore"
"You've Got Mail"	Mike Newell	"Steve Zahn"
"You've Got Mail"	Mike Newell	"Ding Dong"
"You've Got Mail"	Mike Newell	"Parker Posey"
"You've Got Mail"	Mike Newell	"Tom Hanks"
"Top Gun"	Marty Grammer	"Tom Cruise"

# 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.

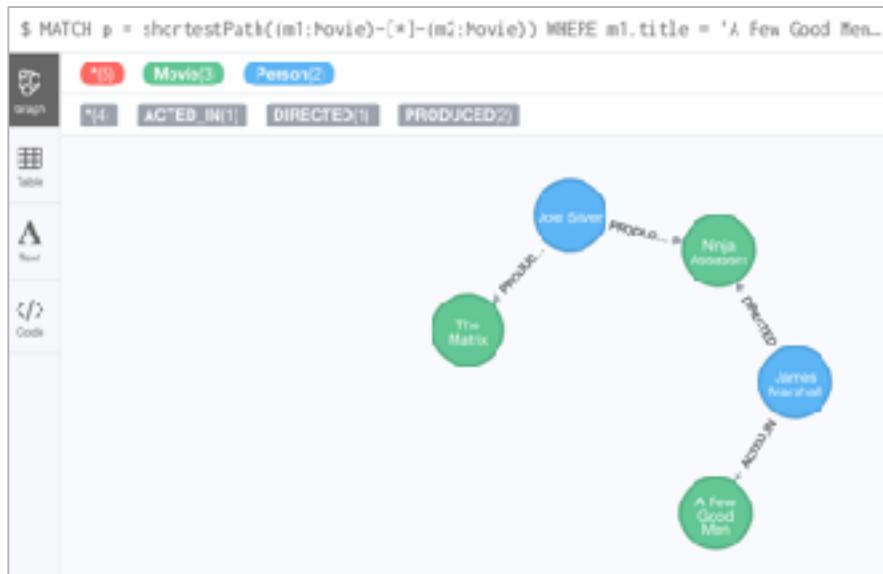
```
MATCH megPath = (meg:Person)-[:ACTED_IN]->(m:Movie)<-
[:DIRECTED]-(:Person),
      (:Person)-[:ACTED_IN]->(m)
WHERE meg.name = 'Meg Ryan'
RETURN megPath
```



# 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.

# 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(*)
```

name	name	count
'Tom Hanks'	'Perry Marshall'	1
'Keanu Reeves'	'Lana Wachowski'	1
'Val Kilmer'	'Tom Scott'	1
'Diane Kruger'	'Michael Cimino'	1
'Helen Mirren'	'Warren Beatty'	1
'Audrey Tautou'	'Ron Howard'	1
'Mandy Patinkin'	'Ron Tyson'	1
'Edie Falco'	'James L. Brooks'	1
'Philip Seymour Hoffman'	'Rob Reiner'	1
'Kathy Bates'	'Mike Nichols'	1
'Jeff Bridges'	'Lawrence Gordon'	1
'Laurence Fishburne'	'Lana Wachowski'	2
'Hugo Weaving'	'Lana Wachowski'	4
'Cate Blanchett'	'James Ivory'	1
'Hugo Weaving'	'Warren Beatty'	1
'Philip Seymour Hoffman'	'Mike Nichols'	1
'Mélanie Laurent'	'Robert Altman'	1

Started streaming 179 records after 8 ms and completed after 8 ms

# 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`
```

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



Table

**movies for Tom Cruise**

```
["Jerry Maguire", "Top Gun", "A Few Good Men"]
```



Text

# 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
```

actor.name	director.name	collaborations	movies
"Lori Petty"	"Wesley Marshall"	1	[ "A League of Their Own"]
"Pamela Reed"	"Eric Packer"	1	[ "Queen Betsy"]
"Valarie" "	"Larry Clark"	1	[ "The Room"]
"Gene Hackman"	"Howard Da Silva"	1	[ "The Godfather"]
"Nick Yarris"	"James Jarmusch"	1	[ "Whiplash"]
"Natalie Trundy"	"Wesley Marshall"	1	[ "The Devil's Candy"]
"Lisa Loeb"	"Tommy Lee Jones"	1	[ "The Assassination of Gianni Versace"]
"Gabe Göencöl Al"	"Vance L. Basler"	1	[ "We Got It from Ed Bono"]
"Cormac blouse"	"Wesley Marshall"	1	[ "A Few Good Men"]
"Bill" Hanks"	"Wesley Marshall"	2	[ "The Last King of Scotland", "A Good Day"]
"Laurence fonsire"	"Eric Packer"	2	[ "The Rock", "The Matrix Reloaded", "The Matrix Revolutions"]
"Hugo Weaving"	"Eric Packer"	2	[ "The Rock", "The Matrix Revolutions", "The Matrix Revolutions", "Cloud Atlas"]
"Ugo Molari"	"Cameron Orme"	1	[ "Every Major"]
"Ugo Molari"	"Vance L. Basler"	1	[ "We Got It from Ed"]
"Philip Seymour Hoffman"	"Mike Nichols"	1	[ "Catch-22's View"]
"Wenon Hwang"	"Peter Weir"	1	[ "What Dreams May Come"]

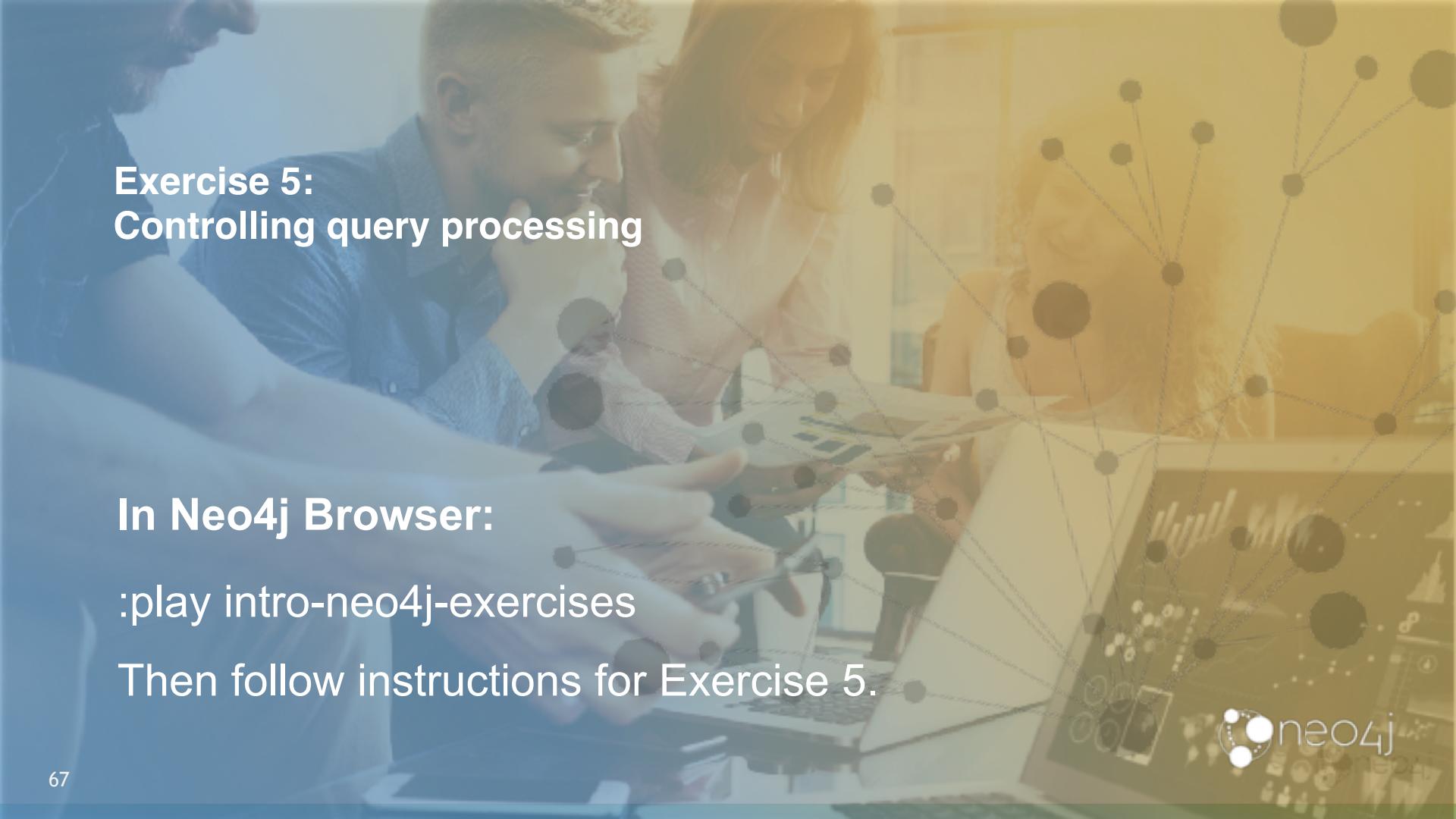
Started streaming 1.5 records after 14 ms and completed after 14 ms.

# 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
```

name	numMovies	movies
"Bill Paxton"	3	["Apollo 13", "Untitled", "A League of Their Own"]
"John Goodman"	3	["Empire of the Sun", "Magical Mystery Tour", "Grand Piano"]
"Peter Fonda"	3	["Grand Piano", "Prairie犬の夢"]
"Helen Hunt"	3	["One Good Thing", "As Good as It Gets", "Gone Away"]
"Gary Oldman"	3	["The Curious Case of Benjamin Button", "Allegory"]
"Morgan Freeman"	3	["The Shawshank Redemption", "The Big Lebowski", "Million Dollar Baby"]
"Steve Buscemi"	3	["The Pianist", "The Black Dahlia", "Lust, Caution"]
"Kathy Bates"	3	["Misery", "The Devil's Own", "The Ballad of Bridget Jones"]
"Catherine O'Hara"	3	["The Fisher King", "The Ballad of Bridget Jones", "The Ballad of Boondock Saints"]
"Sam Rockwell"	3	["Traffic", "Seven Days in Utopia", "The Green Mile"]
"Scoot McNairy"	3	["Traffic", "Seven Days in Utopia"]
"Sanaa Lathan"	3	["The Curious Case of Benjamin Button", "Allegory"]
"Dale Midkiff"	3	["Speed Racer", "Allegory"]
"Rick Roberts"	3	["Unveiling the Captain", "World War II"]
"Mark van Dijken"	3	["Misery", "The Devil's Own", "The Ballad of Bridget Jones"]
"Tina Yothers"	3	["Seven Days in Utopia", "The Curious Case of Benjamin Button"]

A blurred background image shows two people, a man and a woman, sitting at a desk and looking at a laptop screen together. A network graph with nodes and connections is overlaid on the right side of the image.

## **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 - 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
```

released	movies
2012	["Cloud Atlas"]
2006	["The Da Vinci Code"]
2004	["Cast Away"]
1993	["Sleepless in Seattle"]
1994	["That Thing You Do"]
1995	["Joe Versus the Volcano"]
1996	["The Green Mile"]
1998	["You've Got Mail"]
2007	["Charlie Wilson's War"]
2002	["A League of Their Own"]
1995	["Apollo 13"]
2004	["The Polar Express"]

Started streaming 12 records after 1 ms and completed after 1 ms.

# 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
```

released	title
2004	'The Polar Express'
1995	'Apollo 13'
1990	'Cast Away'
1997	'Cloud Atlas'
1999	'The Da Vinci Code'
2006	'Good Will Hunting'
2008	'Inferno'
2000	'The Polar Express'
2012	'Life of Pi'
1993	'The Thin Red Line'
2007	'No Country for Old Men'
2009	'The Da Vinci Code'
1995	'Moneyball'
2013	'Valkyrie'
1994	'What Dreams May Come'
1995	'Young Adult'

Started streaming 15 records after 1 ms and completed after 1 ms.

# 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
```

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

The screenshot shows the Neo4j browser interface with a table result. The table has two columns: 'm.released' and 'movies'. The 'm.released' column lists years from 1980 to 2012. The 'movies' column lists the titles of the movies released in each year. The results are ordered by release year in descending order (from most recent to oldest). The table includes a header row and 13 data rows.

m.released	movies
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"]
1993	["Sleepers in Seattle"]
1992	["A League of Their Own"]
1990	["Joe Versus the Volcano"]

Started streaming 12 records after 2 ms and completed after 2 ms.

# 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
```

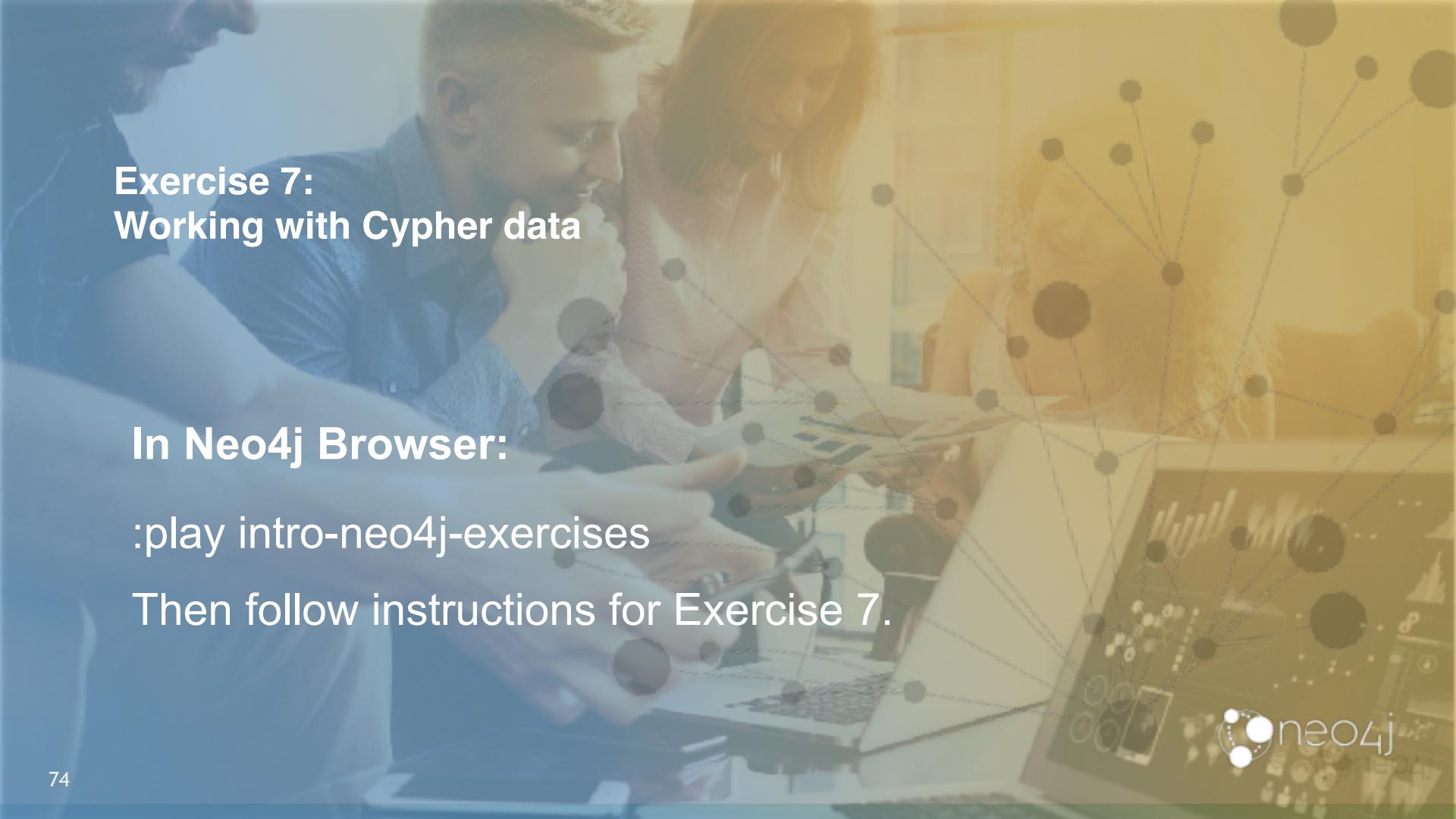
title	year
"Cloud Atlas"	2012
"Ninja Assassin"	2009
"Frost/Nixon"	2008
"Speed Racer"	2008
"Charlie Wilson's War"	2007
"V for Vendetta"	2006
"The Da Vinci Code"	2006
"Rescuers Down Under"	2005
"The Polar Express"	2004
"The Matrix Reloaded"	2003

Started streaming 10 seconds after 1 ms and completed after 2 ms.

# 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, Q4: 509]

A blurred background image shows two people, a man and a woman, sitting at a desk and looking at a computer screen together. A network graph with nodes and connections is overlaid on the right side of the slide.

## **Exercise 7:** **Working with Cypher data**

**In Neo4j Browser:**

`:play intro-neo4j-exercises`

Then follow instructions for Exercise 7.

# 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.

A photograph of two people, a man and a woman, sitting at a desk in an office environment. They are looking down at a laptop screen, possibly discussing data. A large, semi-transparent network graph with nodes and connections is overlaid on the right side of the image.

## Exercise 11: Merging data in the

In Neo4j Browser:

:play intro-neo4j-exercises

Then follow instructions for Exercise 11.



A background image featuring a person's hands holding and fitting together yellow puzzle pieces. Overlaid on this image is a network graph consisting of numerous dark grey circular nodes connected by thin grey lines, representing data relationships.

# Getting More Out of Neo4j

v 1.0

# 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(*)
```

\$ LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/intro-neo4j/movies_to_load.csv" AS line RETURN count(*)	
 Table	count(*)
 A	3

# 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
```

The screenshot shows the Neo4j browser interface with a single record highlighted. The record is a JSON object representing a movie:

```
{  
    "summary": "Every dream has a  
price.",  
    "country": "USA",  
    "id": 11,  
    "title": "Wall Street",  
    "year": "1987"}  
}
```

Two annotations with yellow arrows point to specific fields:

- An arrow points to the "summary" field with the text: "We need to trim leading spaces for the *tagline* property value".
- An arrow points to the "id" field with the text: "We need to convert to integer for the *released* property value".

At the bottom of the browser window, the status bar displays: "Started streaming 1 records after 246 ms and completed after 346 ms."

# 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)
```

\$ LOAD CSV WITH HEADERS FROM 'http://data.neo4j.com/intro-neo4j/movies\_to\_load.csv' ...

Table	line.id	line.title	toInteger(line.year)	trim(line.summary)
A	"1"	"Wall Street"	1987	"Every dream has a price."
Text	"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 {..})
```



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
```

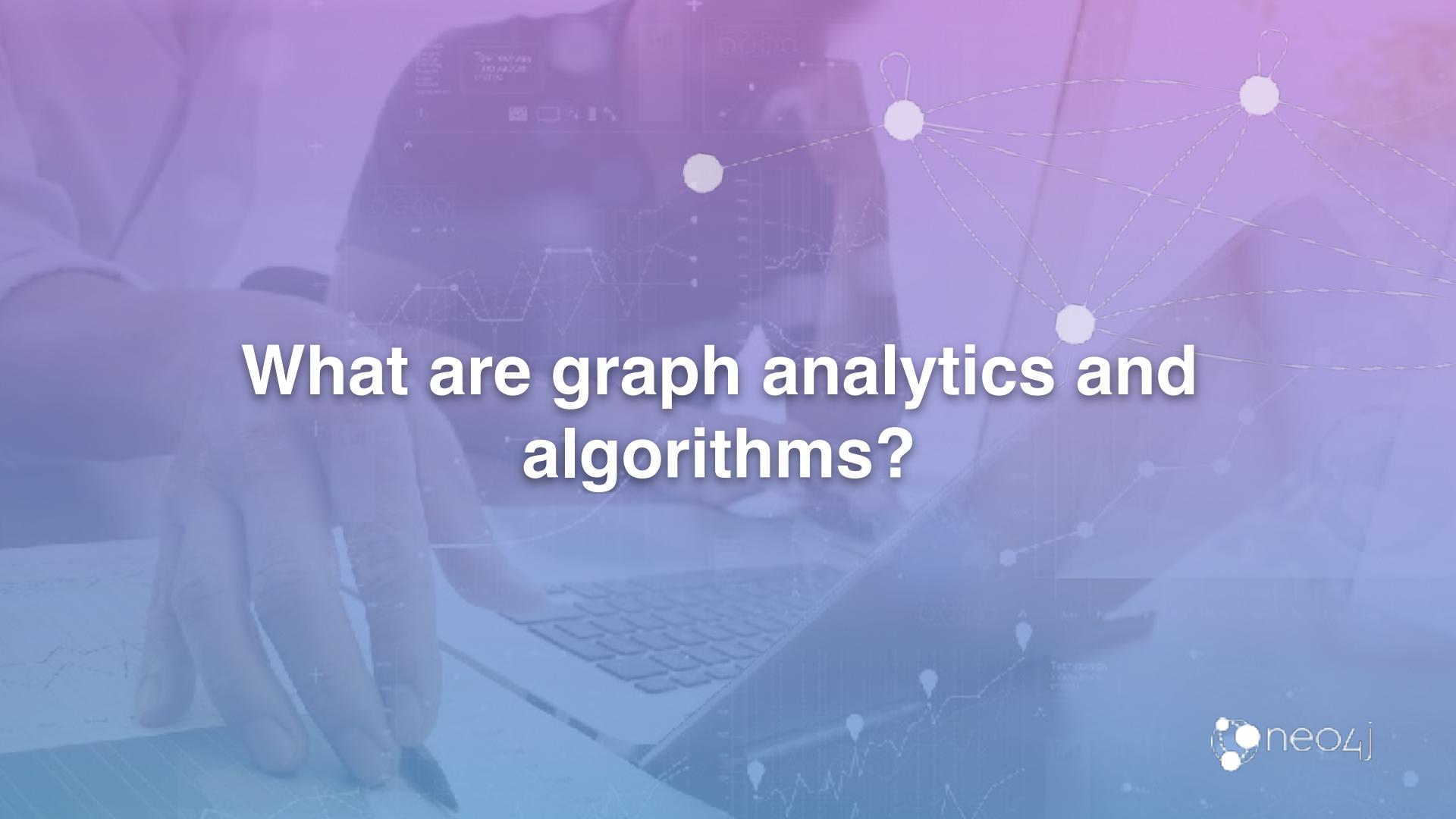
```
LOAD CSV WITH HEADERS
FROM 'https://data.neo4j.com/intro-neo4j/roles_to_load.csv'
AS line
MATCH (movie:Movie { movieId: line.movieId })
MATCH (person:Person { personId: line.personId })
CREATE (person)-[:ACTED_IN { roles: [line.role] }]->(movie)
```

```
$ LOAD CSV WITH HEADERS FROM "https://data.neo4j.com/intro-neo4j/roles_to_load.csv" AS line MATCH (movie:Movie { m...
```



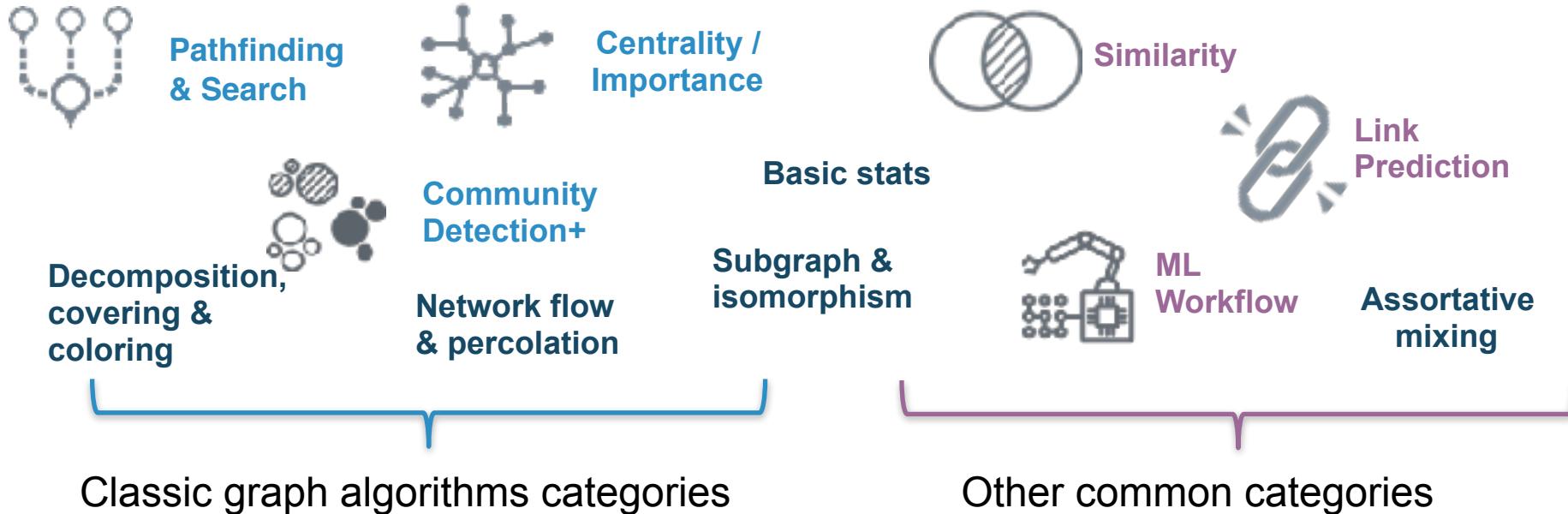
Set 6 properties, created 6 relationships, completed after 323 ms.





# What are graph analytics and algorithms?

# Common types of graph algorithms



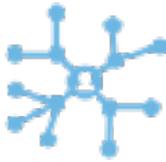
So many others!

# +35 Graph & ML algorithms in Neo4j



## Pathfinding & Search

Finds optimal paths or evaluates route availability and quality.



## Centrality / Importance

Determines the importance of distinct nodes in the network.



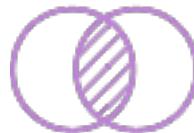
## Community Detection

Detects group clustering or partition options.



Estimates the likelihood of nodes forming a future relationship.

## Link Prediction



## Similarity

Evaluates how alike nodes are.

The book cover features a purple header with the O'Reilly logo and the title 'Graph Algorithms'. Below the title, it says 'Practical Examples in Apache Spark & Neo4j'. The cover art shows a spider in the center of a web, with the Neo4j logo in the top right corner. At the bottom, the authors' names are listed: 'Marc Needham & Amy E. Hodler'.

[neo4j.com/graph-algorithms-book/](http://neo4j.com/graph-algorithms-book/)

# Graph and ML algorithms in Neo4j



## Pathfinding & Search

- Parallel Breadth First Search & Depth First Search
- Shortest Path
- Single-Source Shortest Path
- All Pairs Shortest Path
- Minimum Spanning Tree
- A\* Shortest Path
- Yen's K Shortest Path
- K-Spanning Tree (MST)
- Random Walk



## Centrality / Importance

- Degree Centrality
- Closeness Centrality
- CC Variations: Harmonic, Dangalchev, Wasserman & Faust
- Betweenness Centrality
- Approximate Betweenness Centrality
- PageRank
- Personalized PageRank
- ArticleRank
- Eigenvector Centrality



## Community Detection

- Triangle Count
- Clustering Coefficients
- Connected Components (Union Find)
- Strongly Connected Components
- Label Propagation
- Louvain Modularity – 1 Step & Multi-Step
- Balanced Triad (identification)



## Similarity

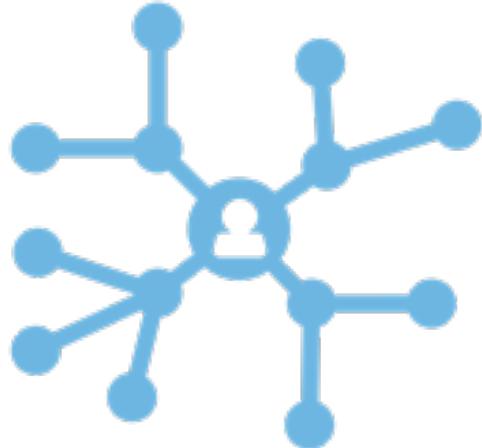
- Euclidean Distance
- Cosine Similarity
- Jaccard Similarity
- Overlap Similarity
- Pearson Similarity
- Approximate Nearest Neighbors



## Link Prediction

- Adamic Adar
- Common Neighbors
- Preferential Attachment
- Resource Allocations
- Same Community
- Total Neighbors

# Centrality algorithms



Determines the importance of distinct nodes in the network.

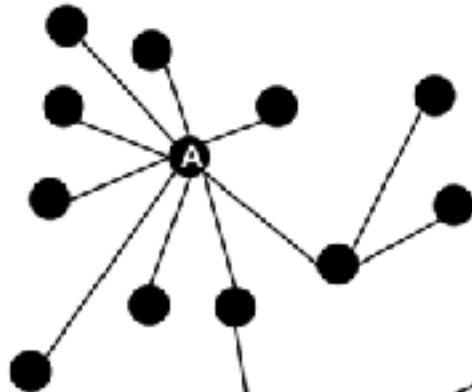
Developed for distinct uses or types of importance.

# Some Centrality algorithms

## Degree

Number of connections?

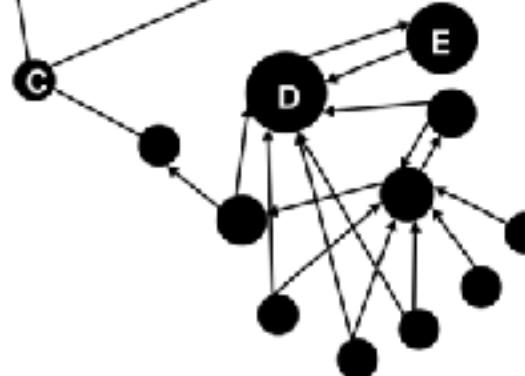
"A" has a high degree



## Betweenness

Which node has the most control over flow between nodes and groups?

"C" is a bridge



## Closeness

Which node can most easily reach all other nodes in a graph or subgraph?

"B" is closest with the fewest hops in its subgraph

## PageRank

Which node is the most important?

"D" is foremost based on number & weighting of in-links

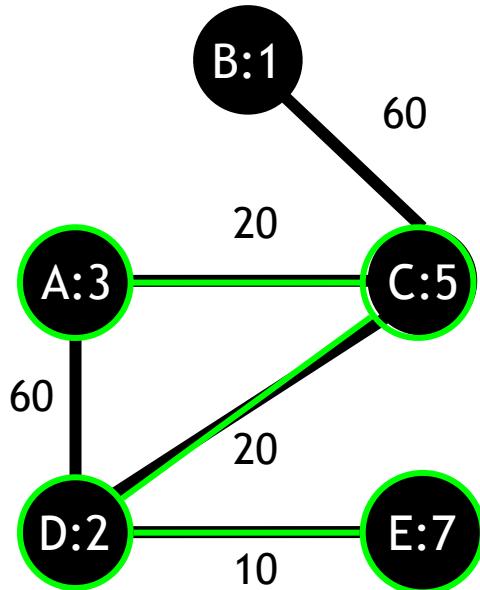
"E" is next, due to the influence of D's link

# Pathfinding and Graph Search algorithms

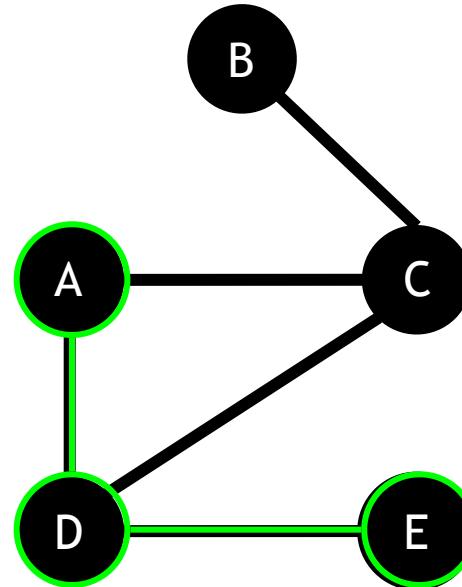


Pathfinding and Graph Search algorithms are used to identify optimal routes, and they are often a required first step for many other types of analysis.

# Shortest Path

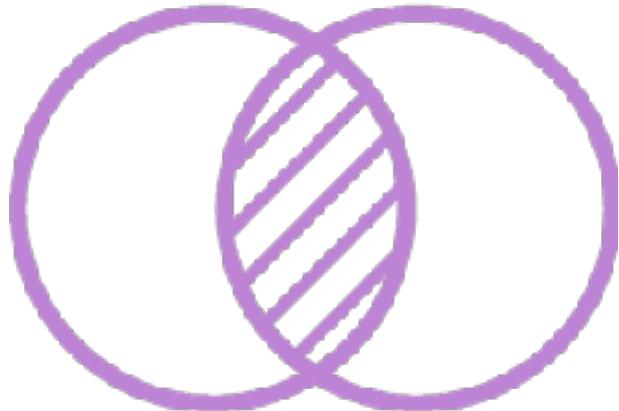


Weighted



Unweighted

# Similarity Algorithms

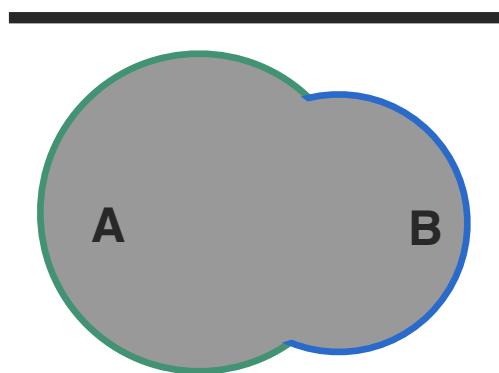
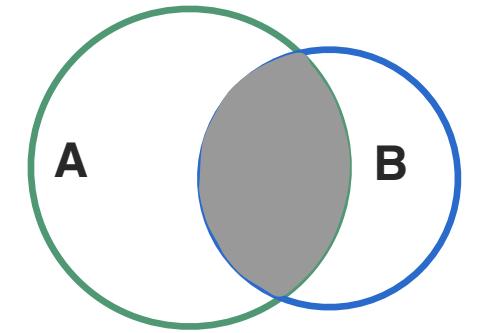


Evaluates how alike nodes are at an individual level either based on node attributes, neighboring nodes, or relationship properties

# Jaccard Similarity coefficient

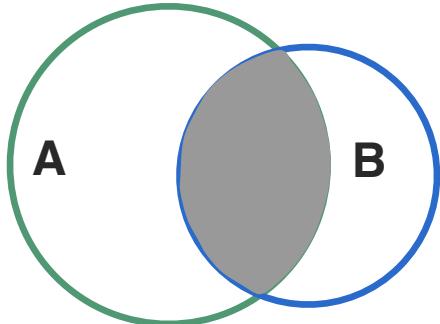
Often used to find recommendations of similar items as well as part of link prediction.

Jaccard Similarity measures the similarity between sets.



$$\frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

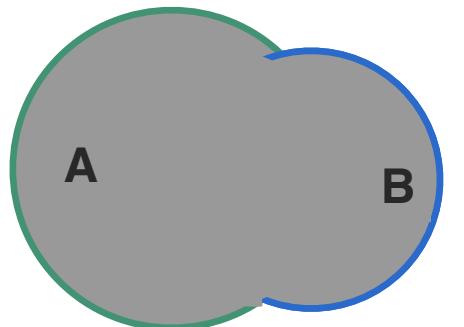
# Jaccard Similarity coefficient



$A = \{\text{Orange, Banana, Cherry, Pineapple}\}$

$B = \{\text{Orange, Banana, Apple}\}$

$$|A \cap B| = |\{\text{Orange, Banana}\}| = 2$$



$$|A \cup B| = |A| + |B| - |A \cap B|$$

$$|A \cup B| = |\{\text{Orange, Banana, Cherry, Pineapple, Apple}\}| = 5$$

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = 2/5 = 0.4$$

# 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/>.

# Further questions just email!

[justin.fine@neo4j.com](mailto:justin.fine@neo4j.com)

Field Engineer



neo4j