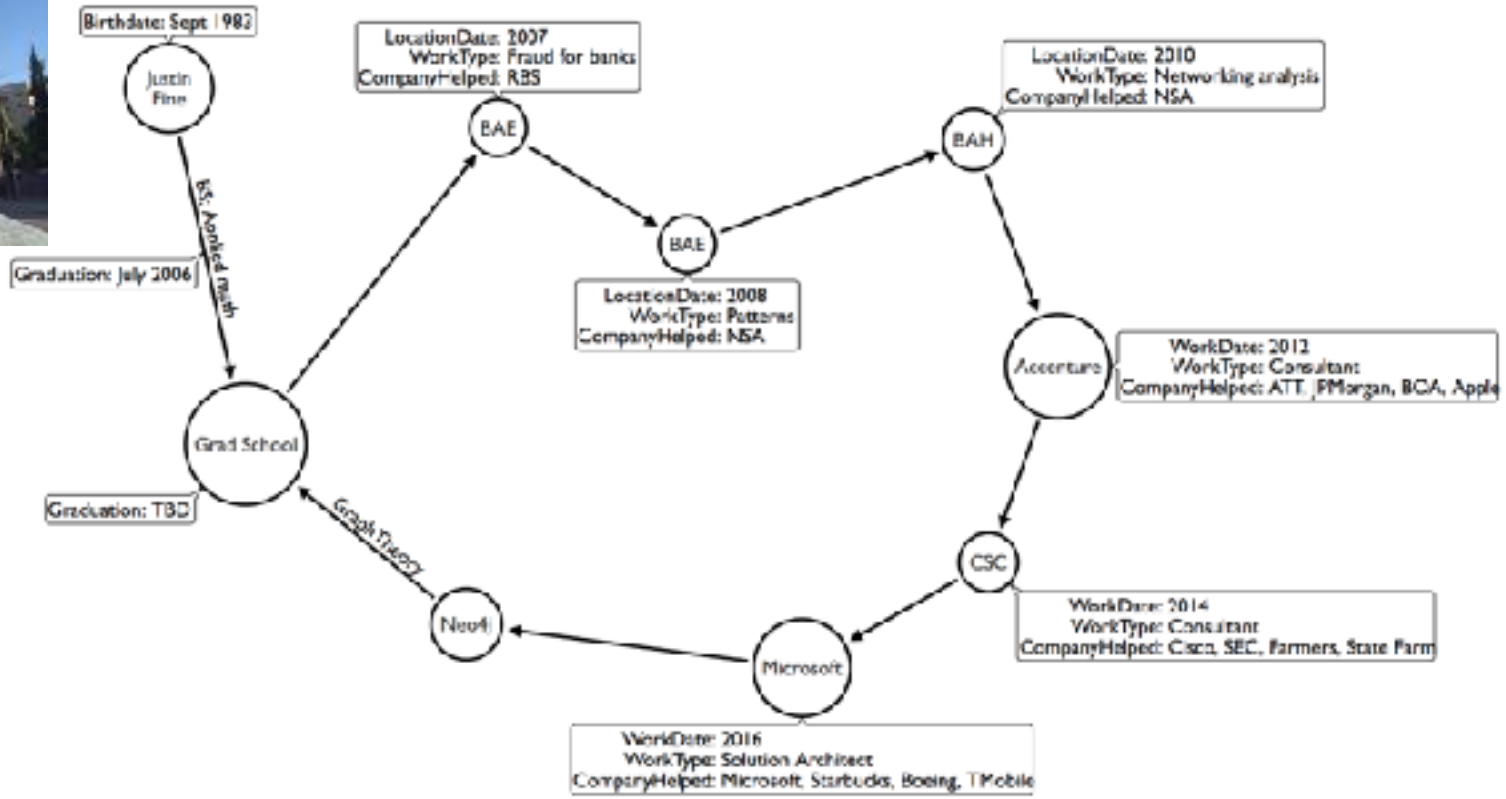

Introduction to
Graph Databases
neo4j
v 1.0


Birthdate: Sept 1982
Justin Fino
BS: Applied Math
Graduation: July 2006
Grad School
Graduation: TBD
LocationDate: 2007
WorkType: Fraud for banks
CompanyHelped: RBS
BAE
BAE
LocationDate: 2008
WorkType: Patterns
CompanyHelped: NSA
LocationDate: 2010
WorkType: Networking analysis
CompanyHelped: NSA
BAH
Accenture
WorkDate: 2012
WorkType: Consultant
CompanyHelped: ATT, JPMorgan, BOA, Apple
CSC
WorkDate: 2014
WorkType: Consultant
CompanyHelped: Cisco, SEC, Farmers, State Farm
Microsoft
WorkDate: 2016
WorkType: Solution Architect
CompanyHelped: Microsoft, Starbucks, Boeing, TMobile
Neo4j
Graph Theory

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.

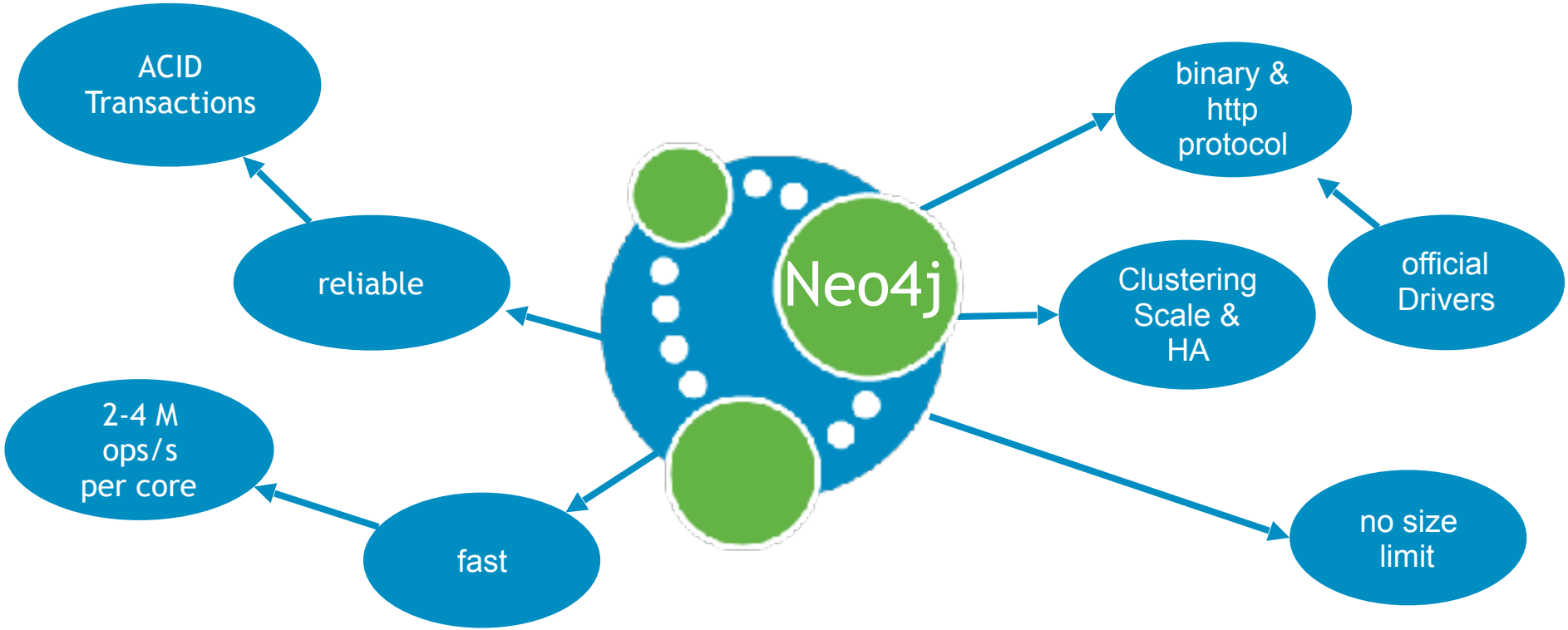
Lingo

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

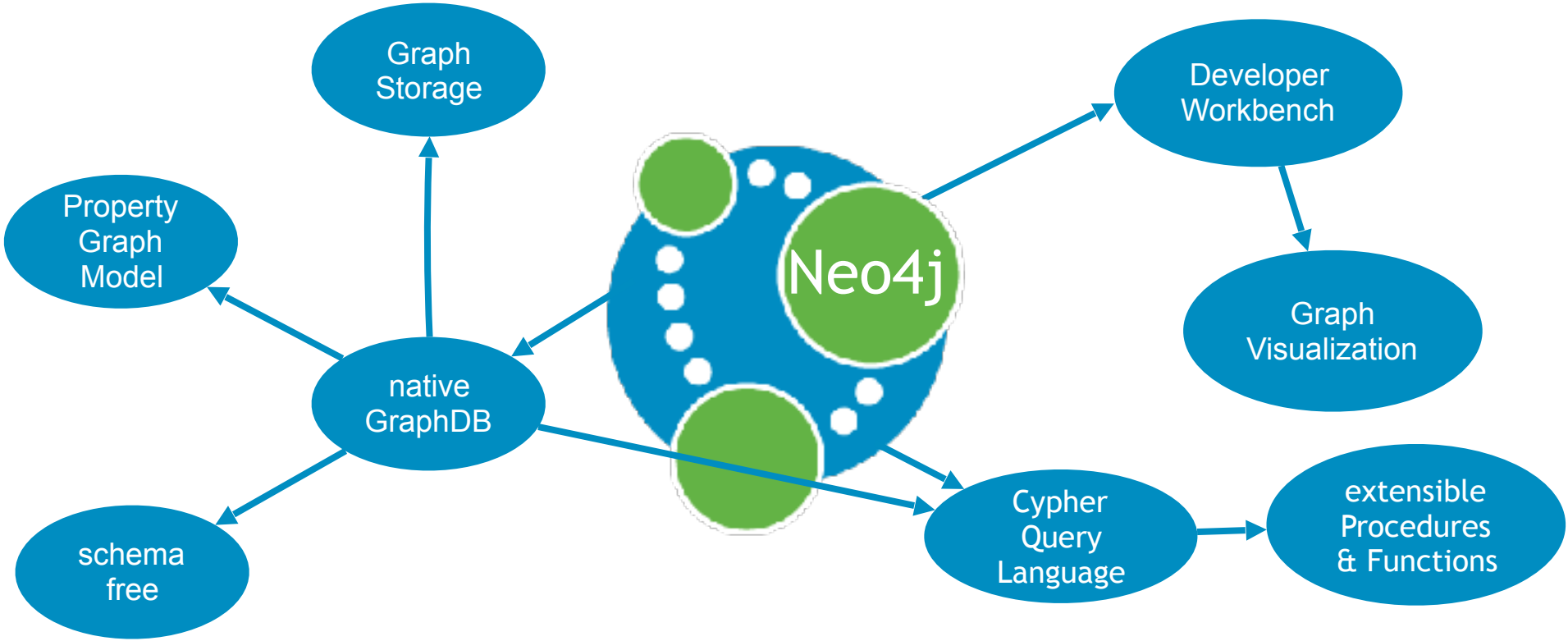
Cool Graph Background

- 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

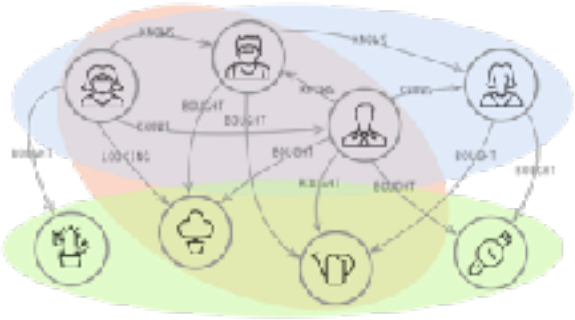
Use cases

Internal Applications

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

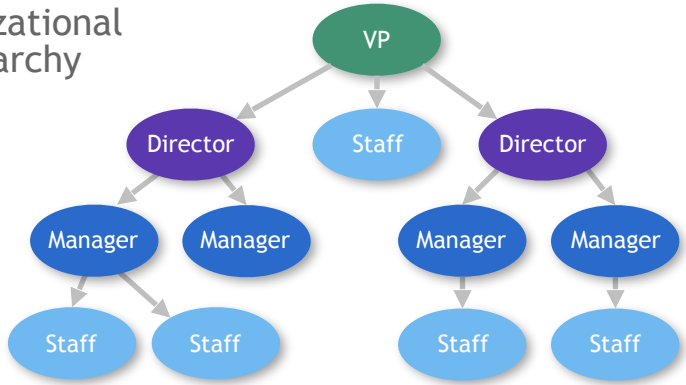
Customer-Facing Applications

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



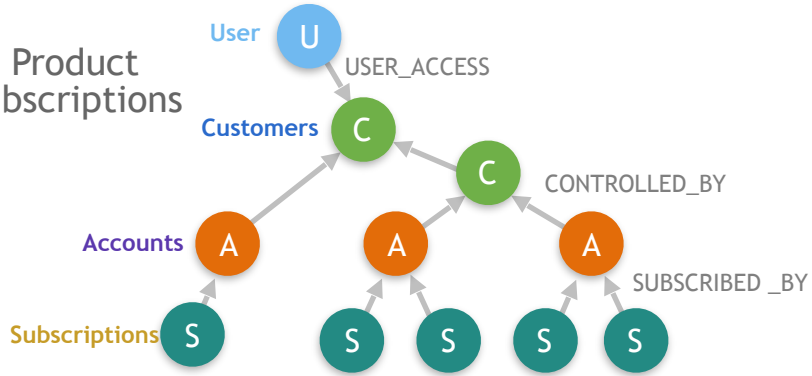
Use cases: Master data management.

Organizational Hierarchy

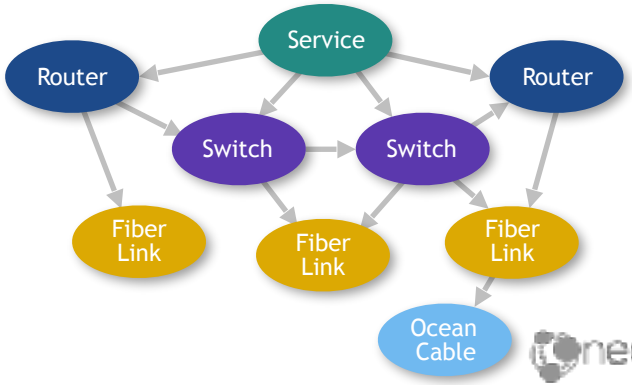


Customer 360

Product Subscriptions



CMDB Network Inventory



What is a graph?

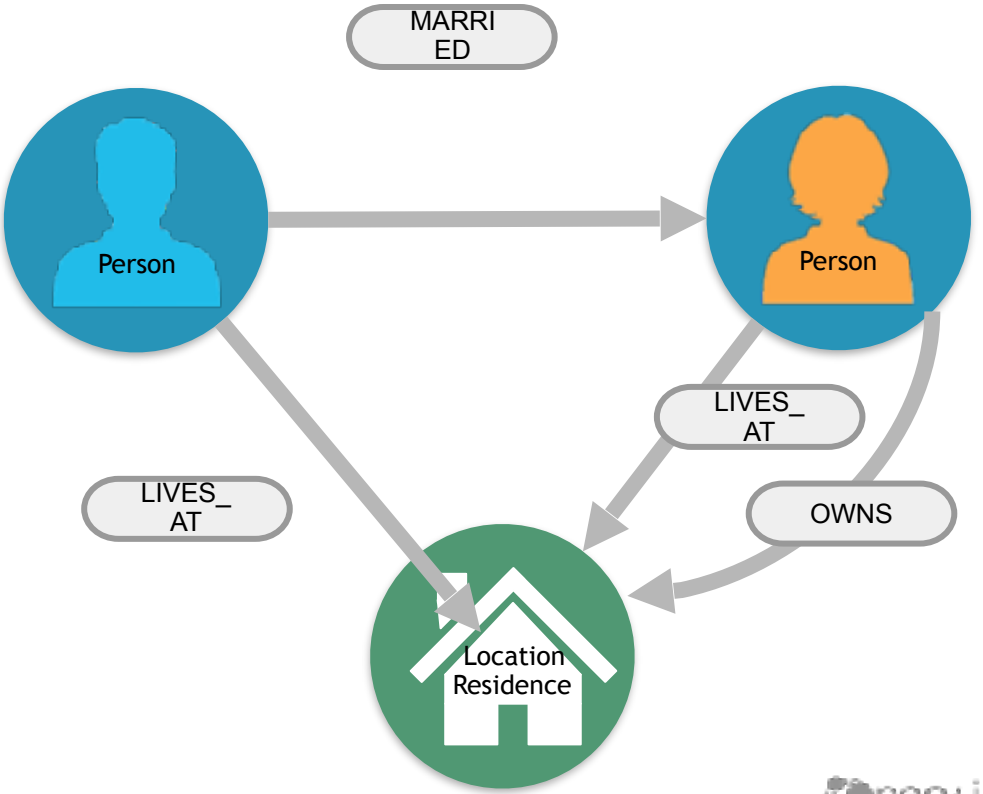
- Nodes
- Relationships
- Properties
- Labels

Nodes

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

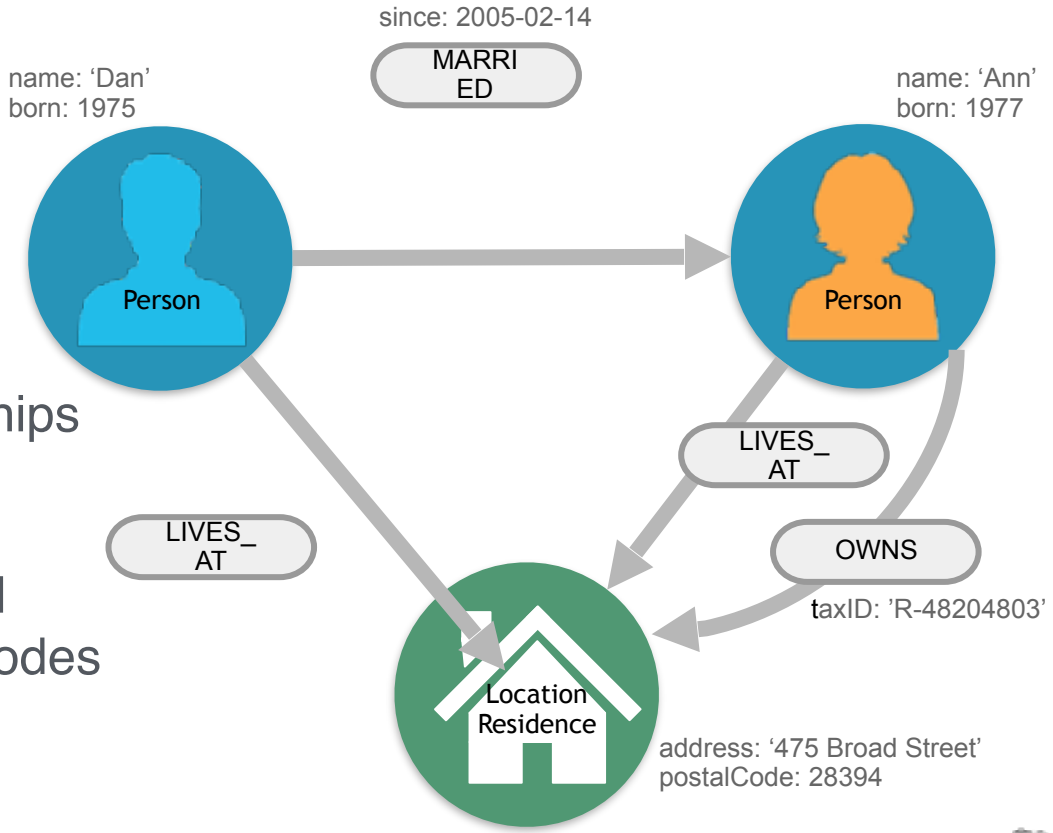
Relationships

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

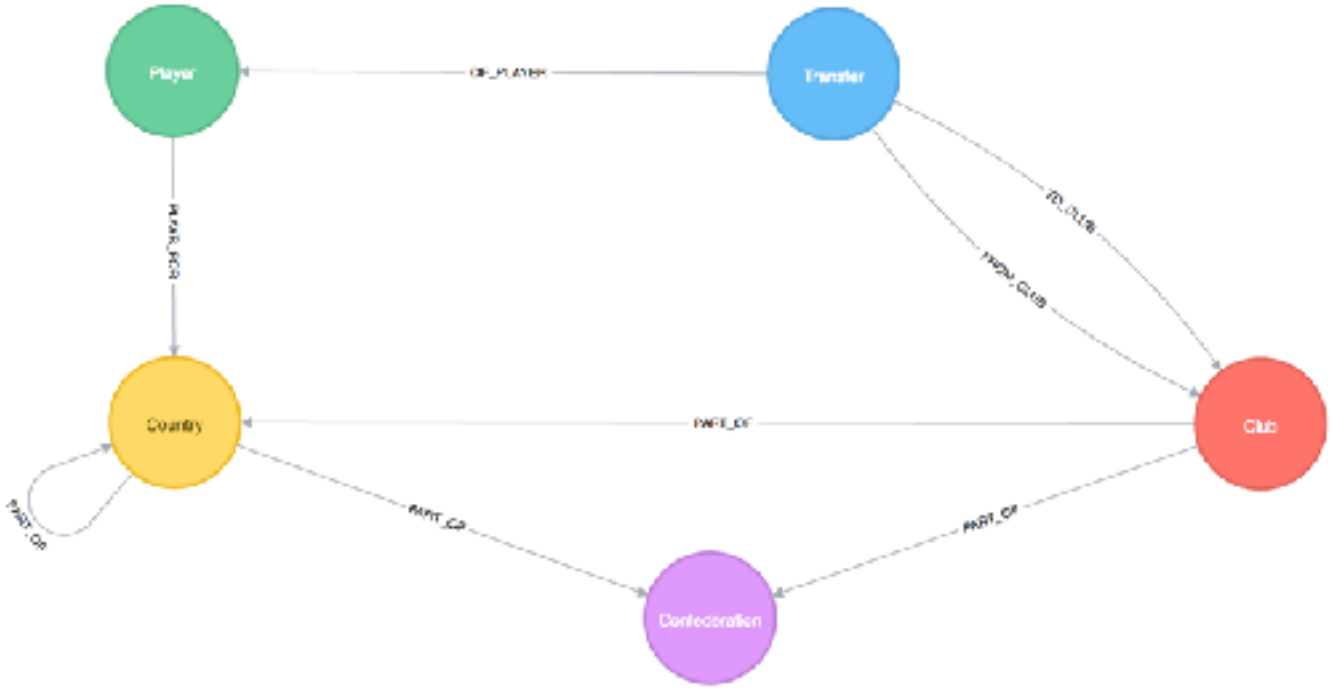


Properties

- 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
neo4j

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:

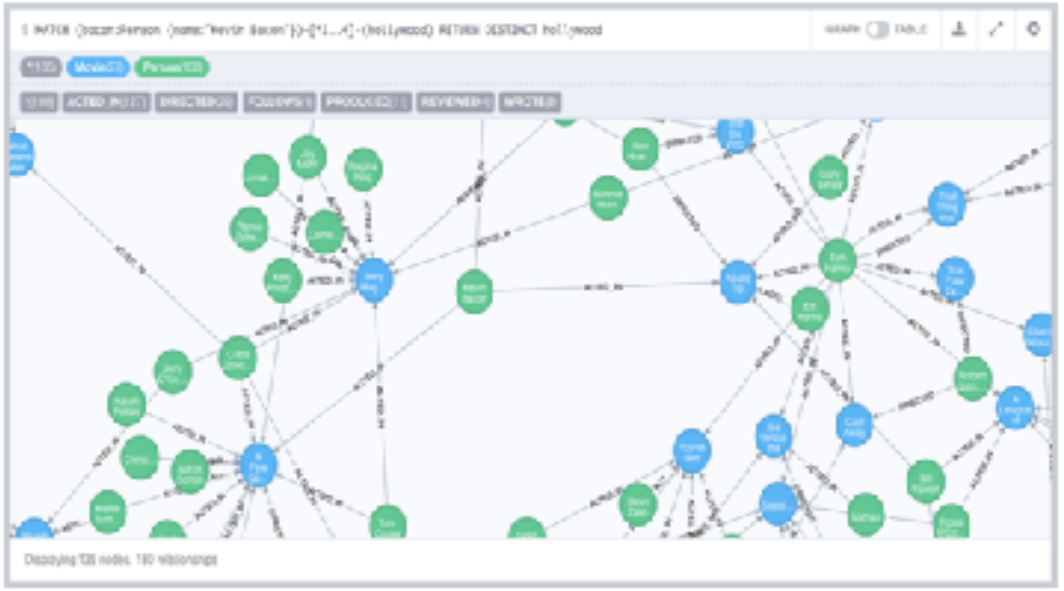
- Awesome Procedures on Cypher (APOC)
- Graph Algorithms
- GraphQL

Neo4j community has contributed many specialized libraries also.

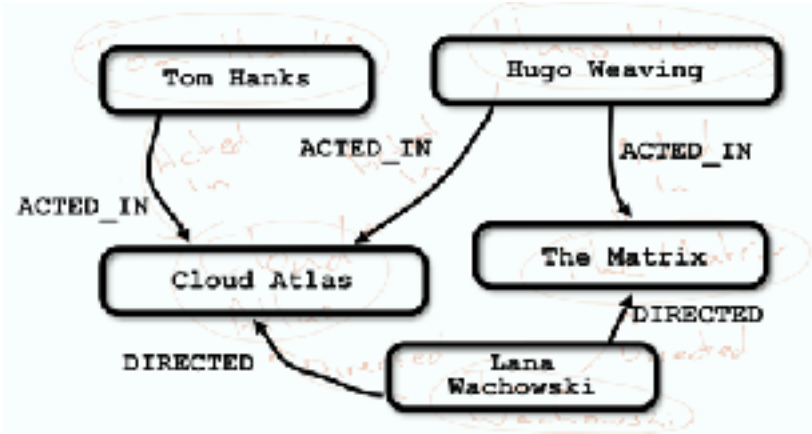
Tools

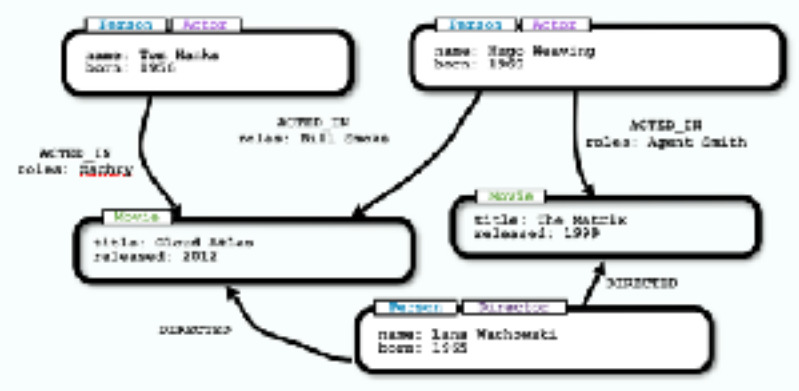
- Neo4j Desktop
- Neo4j Browser
- Neo4j Bloom
- Neo4j ETL Tool

Neo4j community has contributed many specialized tools also.

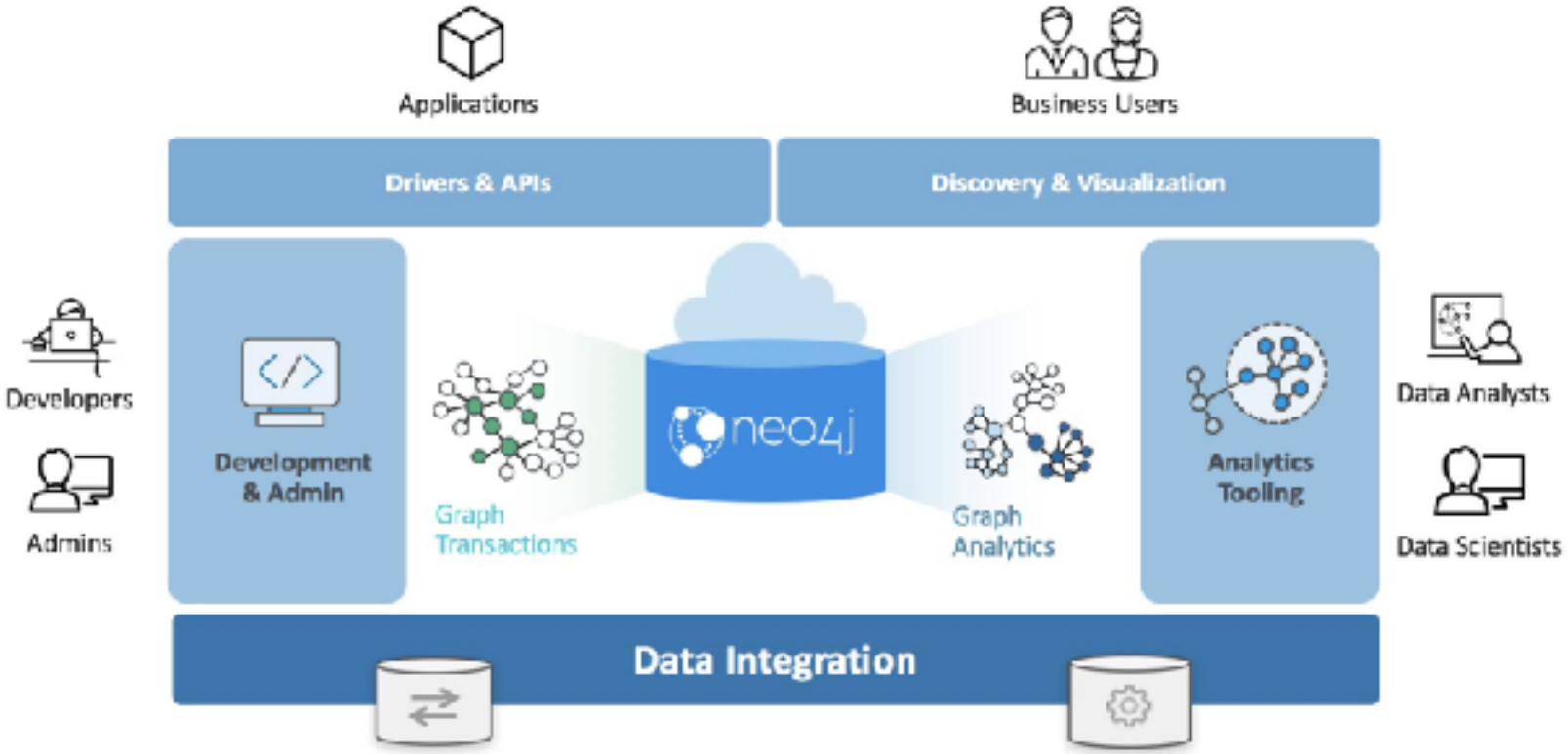


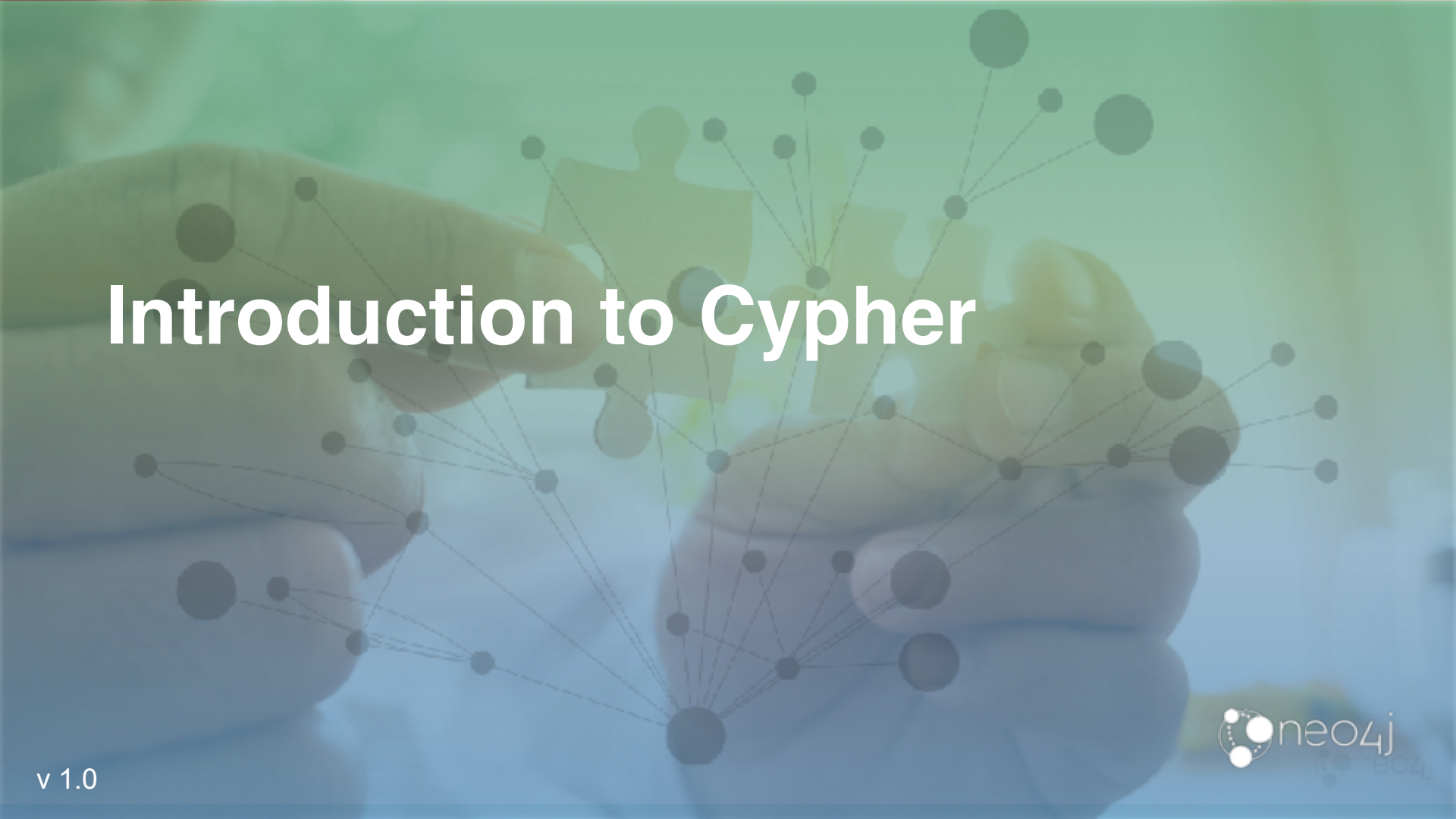
Whiteboard modeling





Neo4j Graph Platform architecture




Introduction to Cypher
neo4j
v 1.0

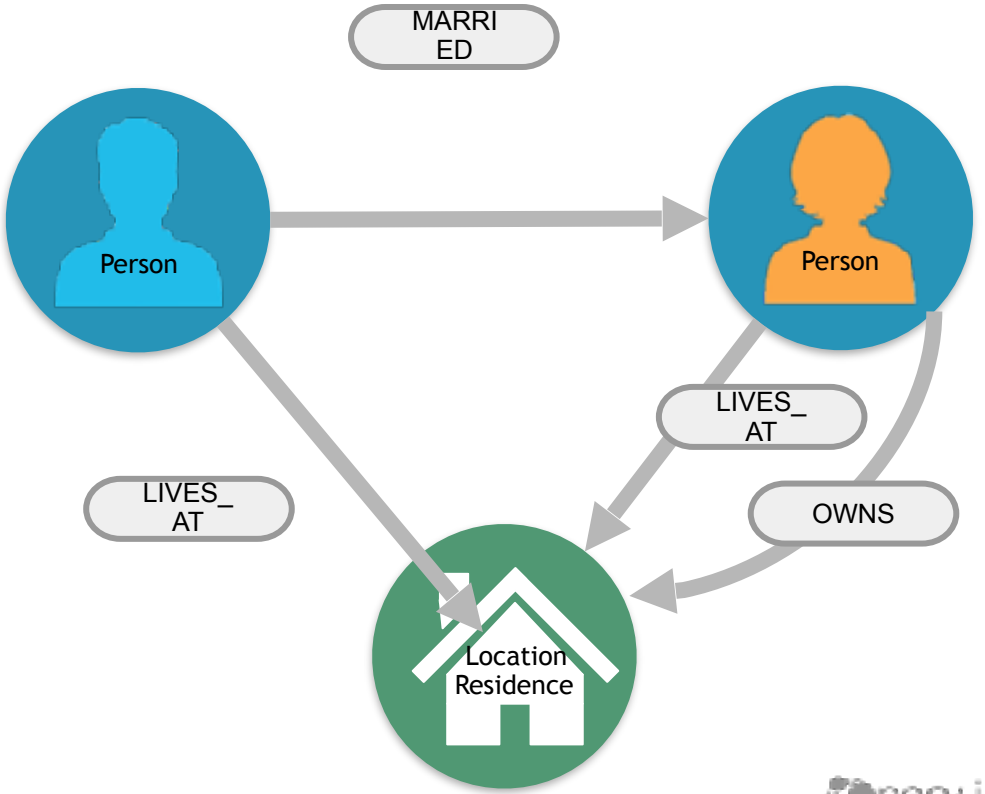
Overview

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

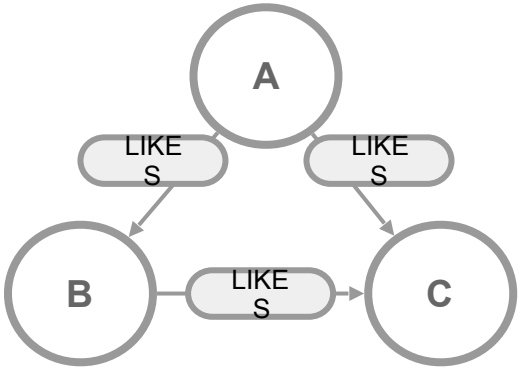
- Retrieve nodes from the graph.
- Filter nodes retrieved using labels and property values of nodes.
- Retrieve property values from nodes in the graph.
- Filter nodes retrieved using relationships.

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

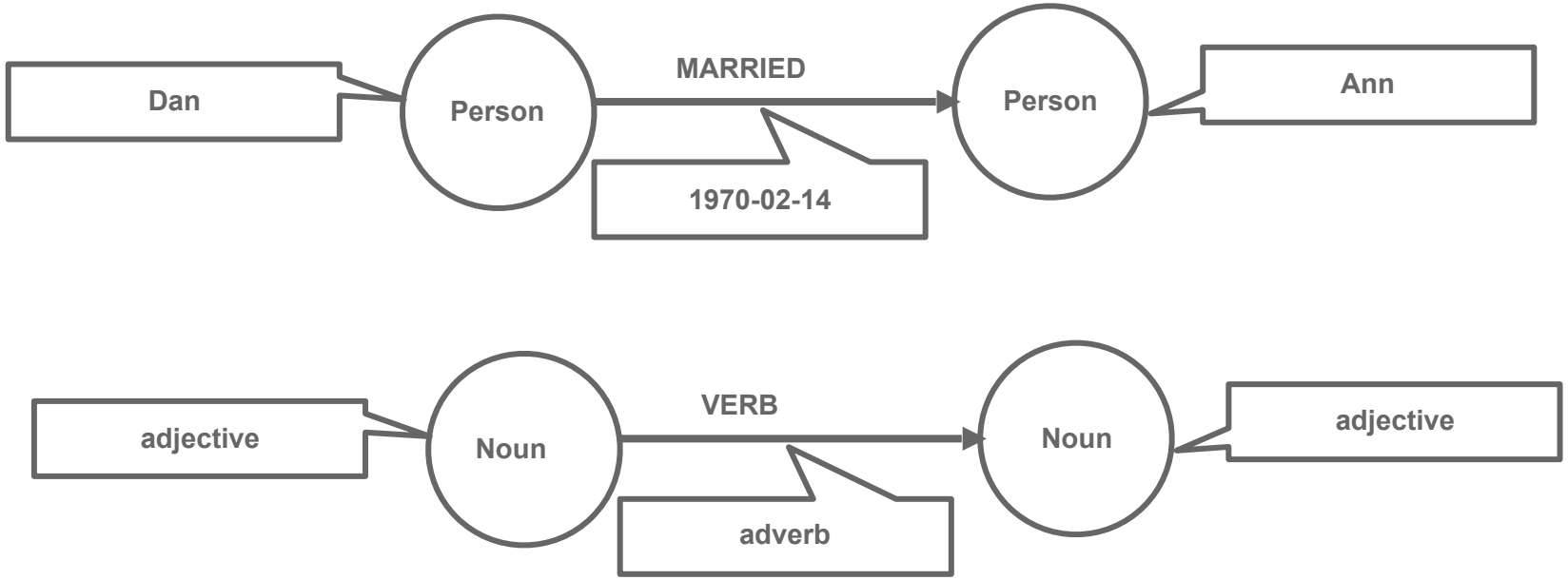


Cypher is ASCII Art



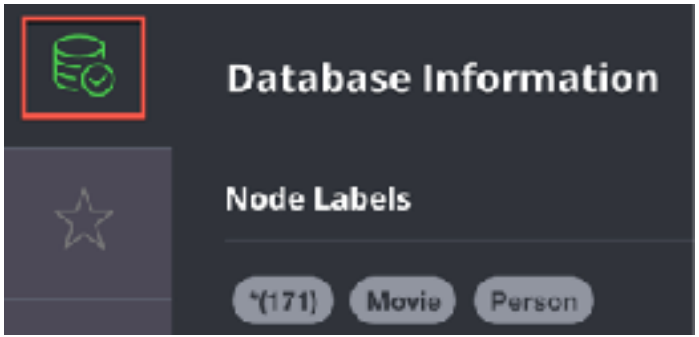

```
(A)-[:LIKES]->(B),(A)-[:LIKES]->(C),(B)-[:LIKES]->(C)

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

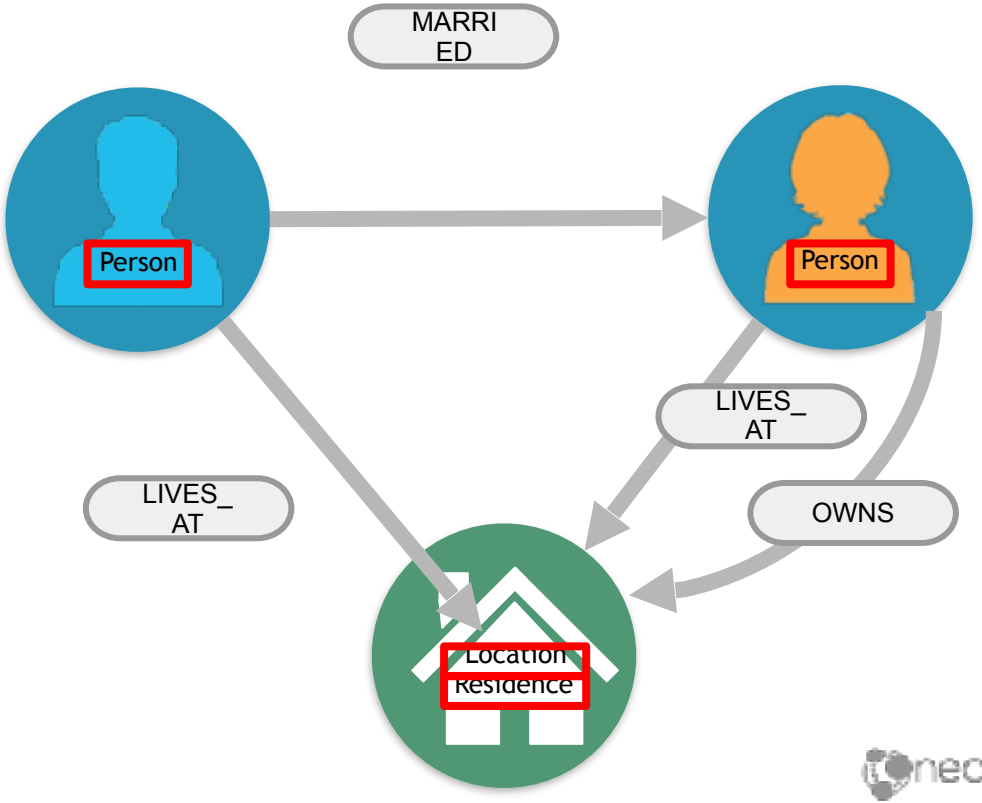
Cypher is readable



Labels

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





Comments in Cypher

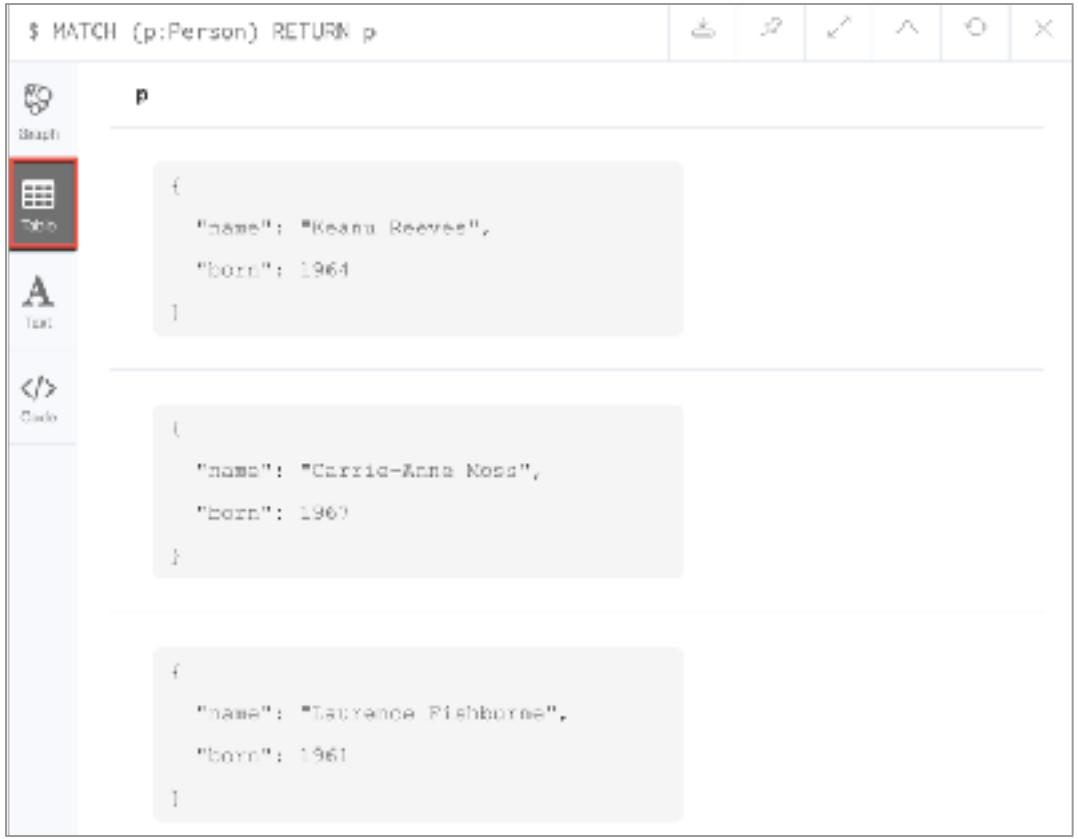
```
()                    // anonymous node not be referenced later in the query
(p)                   // variable p, a reference to a node used later
(:Person)             // anonymous node of type Person
(p:Person)            // p, a reference to a node of type Person
(p:Actor:Director)    // p, a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

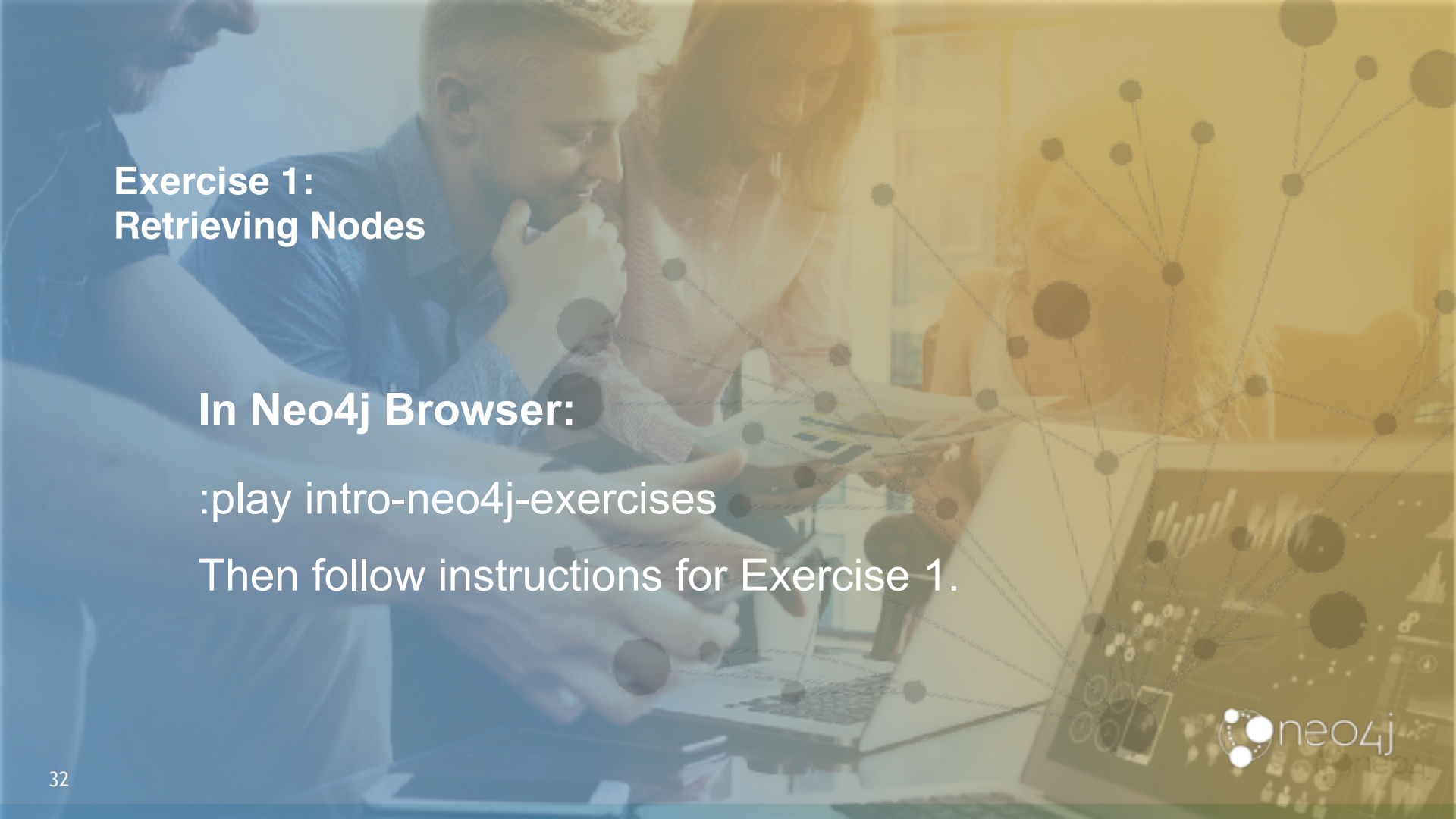
Using MATCH to retrieve nodes

```
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





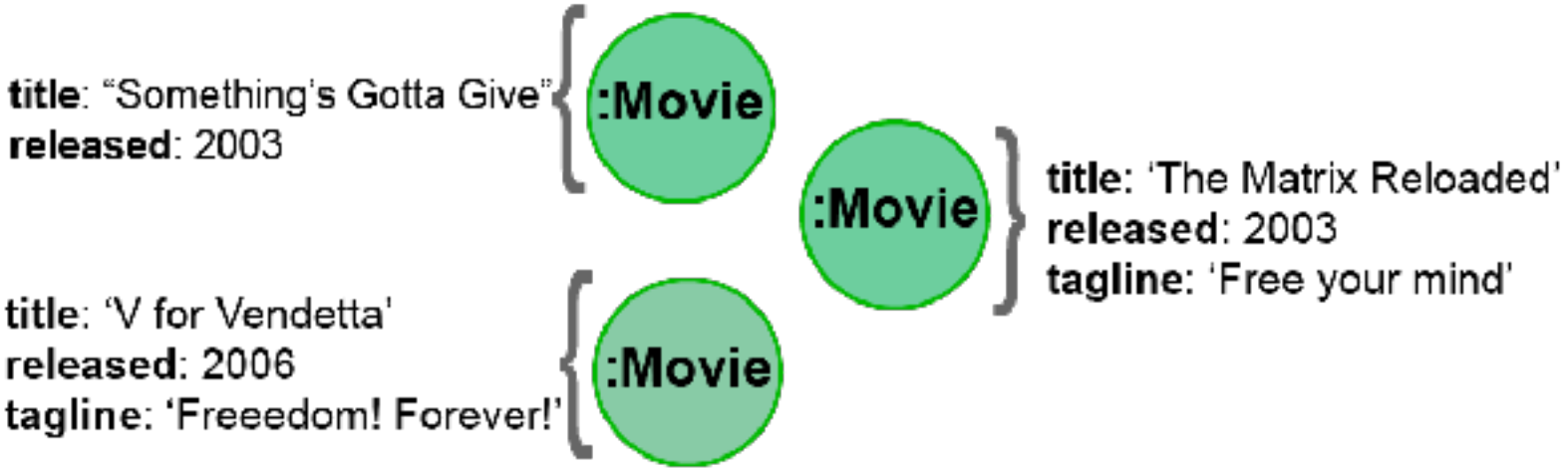
Exercise 1:
Retrieving Nodes

In Neo4j Browser:

:play intro-neo4j-exercises

Then follow instructions for Exercise 1.

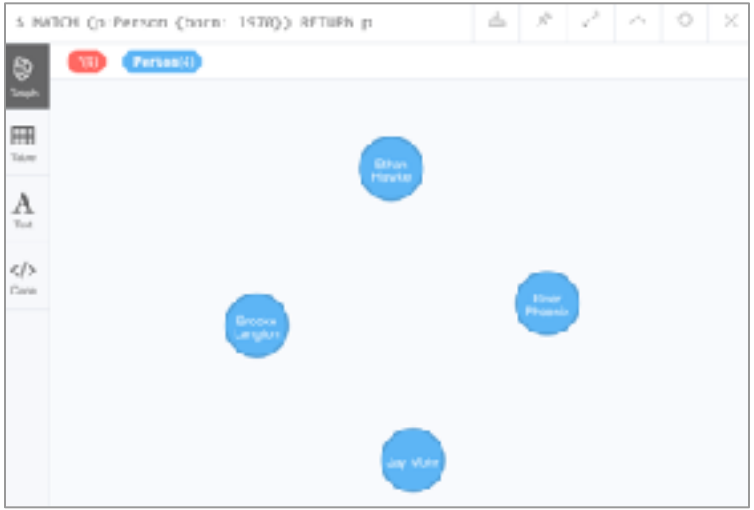
Properties


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

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,

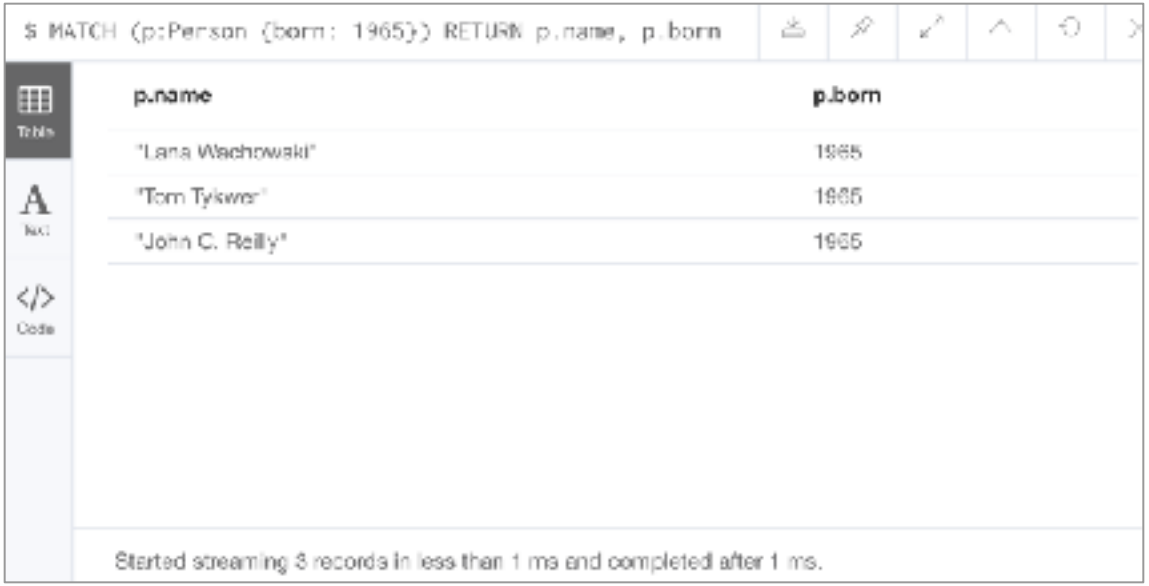
Free your mind, returning the nodes:

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

Returning property values

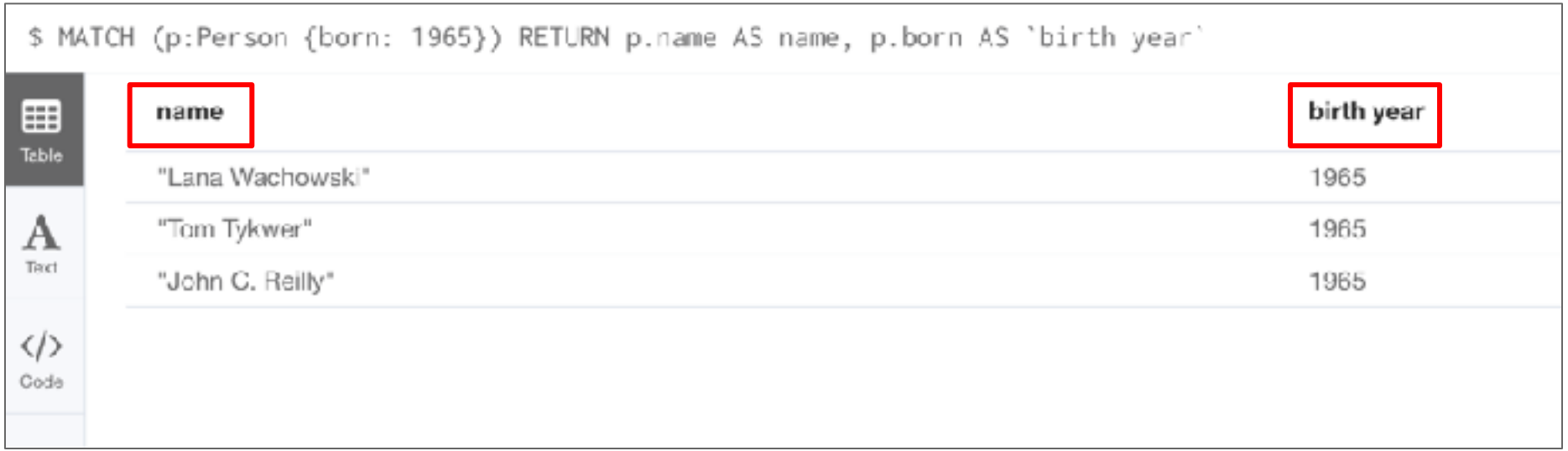
Find all people born in *1965* and return their names:

```
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'
```



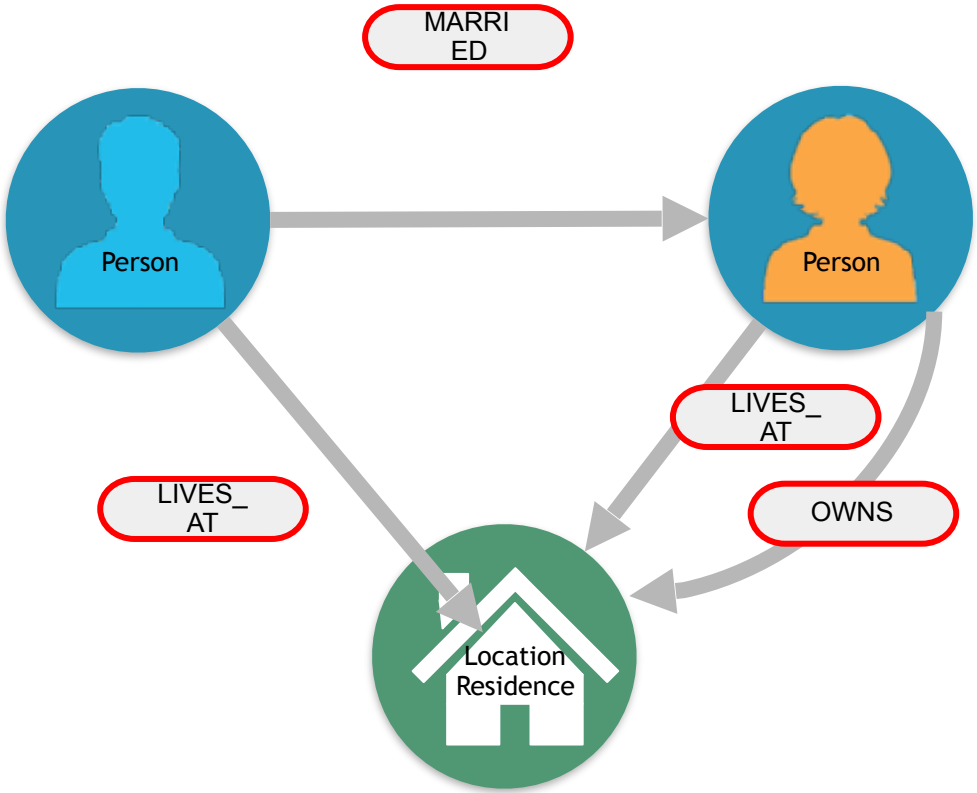
Exercise 2:
Filtering queries using property values

In Neo4j Browser:

:play intro-neo4j-exercises

Then follow instructions for Exercise 2.

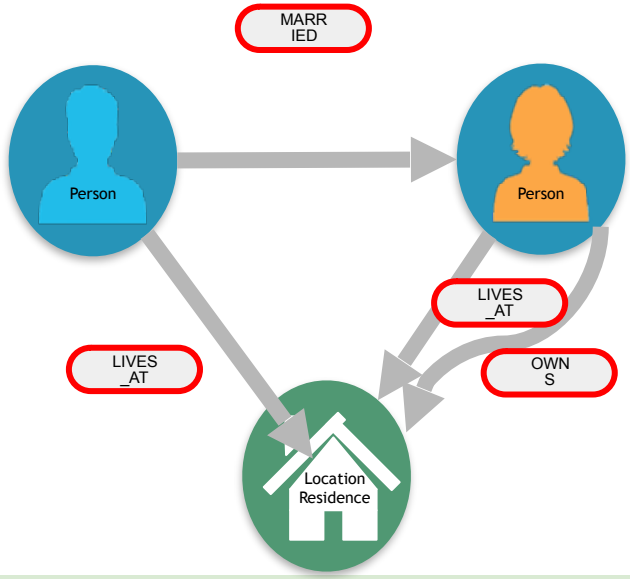
Relationships



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
```

Querying by multiple relationships

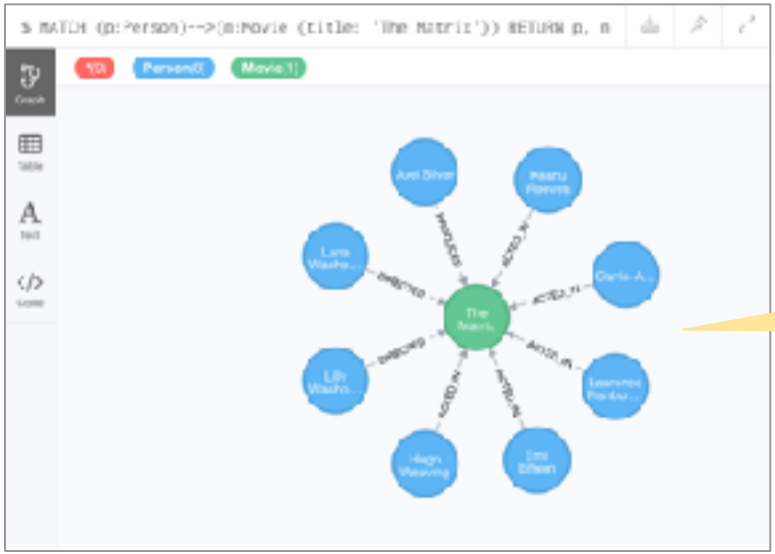
Find all movies that *Tom Hanks* acted in or directed and return the title of the move:

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

Using 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
```

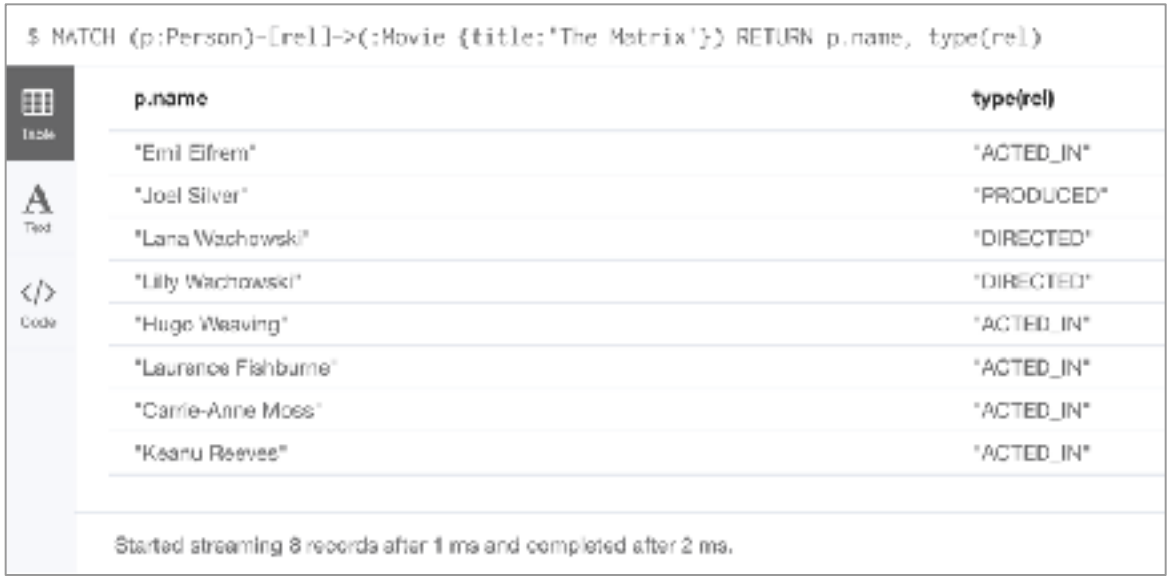


Connect result nodes enabled in Neo4j Browser

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)
```

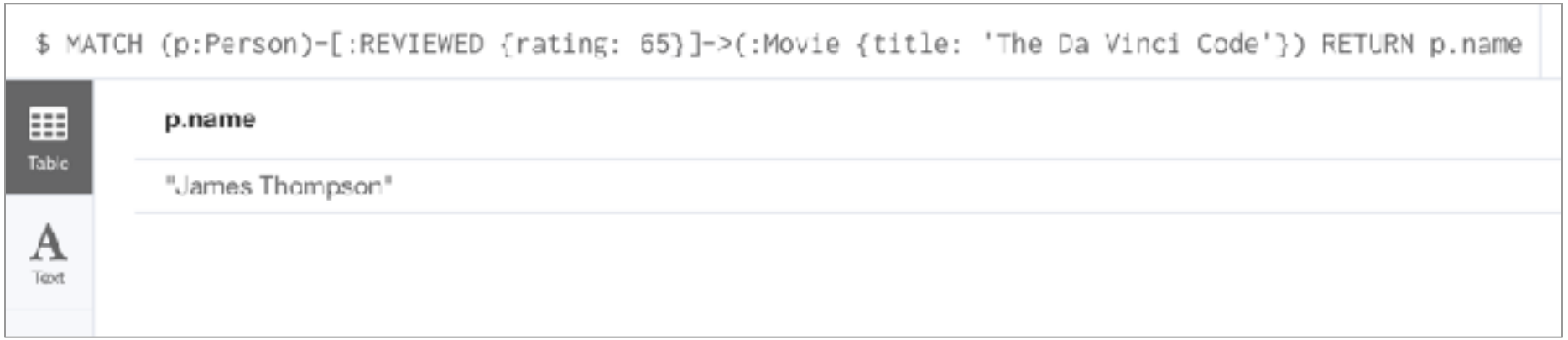


Retrieving properties for a relationship - 1

Retrieving properties for a relationship - 2

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

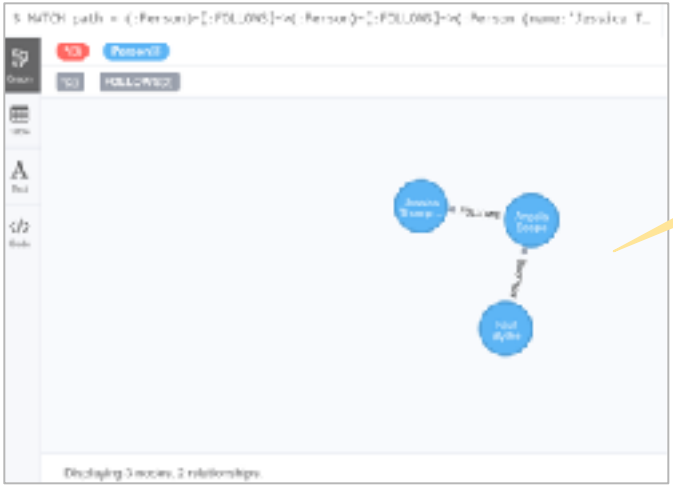
```
MATCH (p:Person)-[:REVIEWED {rating: 65}]->(:Movie {title: 'The Da Vinci Code'})
RETURN p.name
```



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


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:

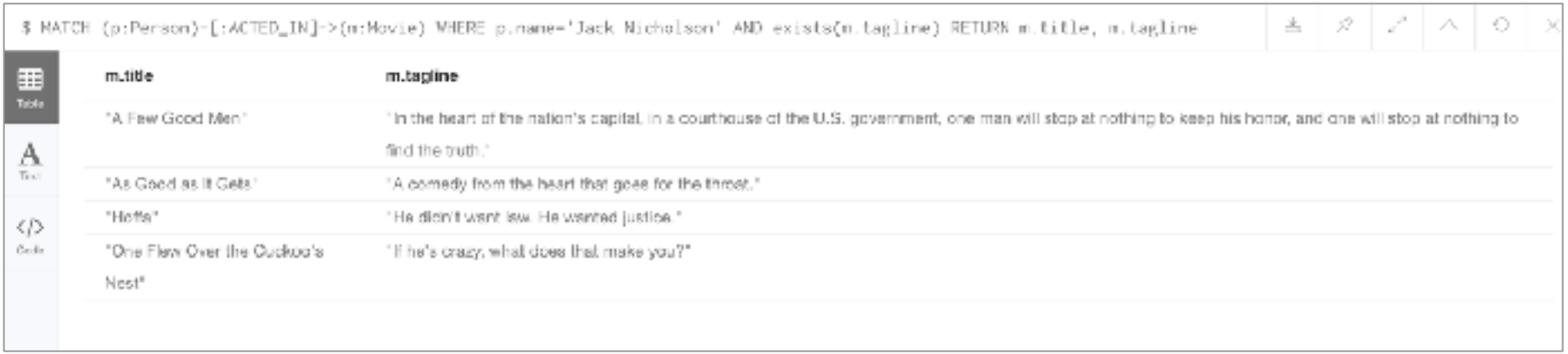
```
MATCH (p)
WHERE p:Person
RETURN p.name
```

```
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:

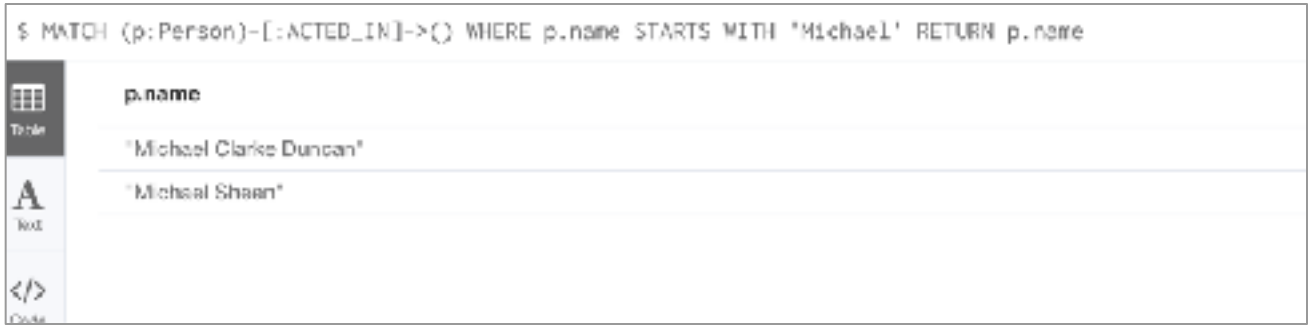
```
MATCH (p:Person)-[:ACTED_IN]->(m:Movie)
WHERE p.name='Jack Nicholson' AND exists(m.tagline)
RETURN m.title, m.tagline
```



Testing strings

Find all actors whose name begins with *Michael*:

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

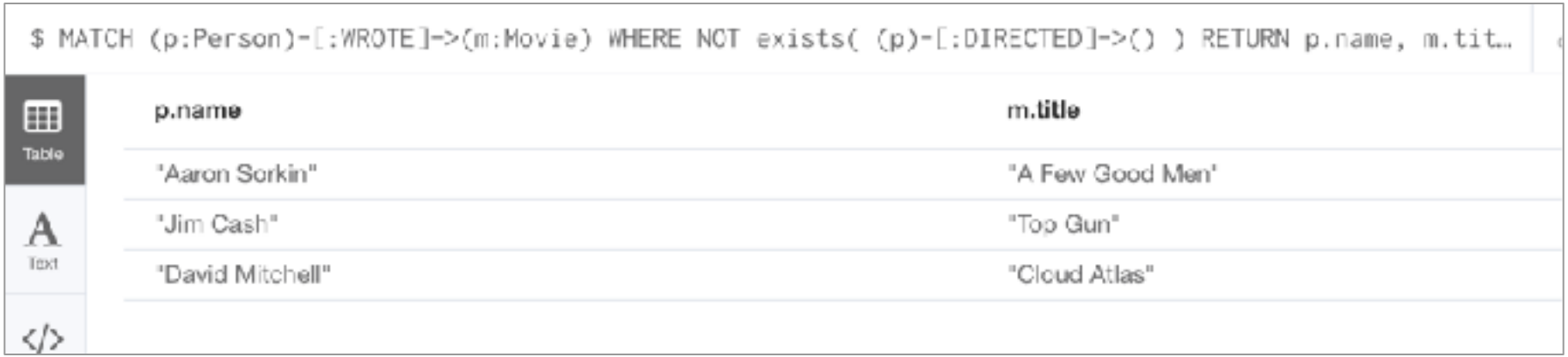



```
MATCH (p:Person)-[:ACTED_IN]->()
WHERE toLower(p.name) STARTS WITH 'michael'
RETURN p.name
```

Testing with patterns - 2

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

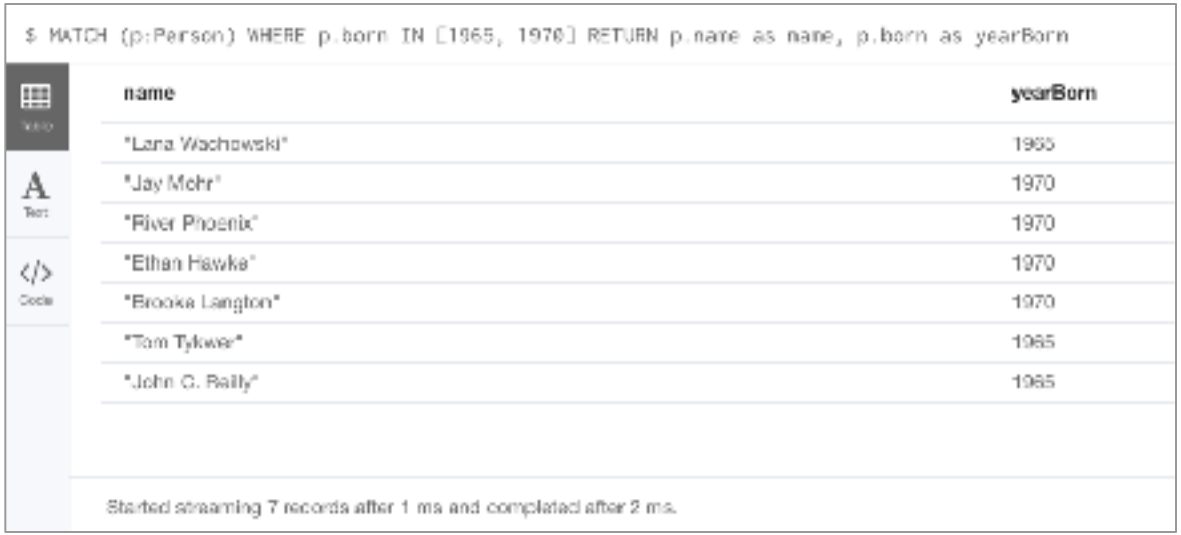
```
MATCH (p:Person)-[:WROTE]->(m:Movie)
WHERE NOT exists( (p)-[:DIRECTED]->() )
RETURN p.name, m.title
```



Testing with list values - 1

Find all people born in 1965 or 1970:

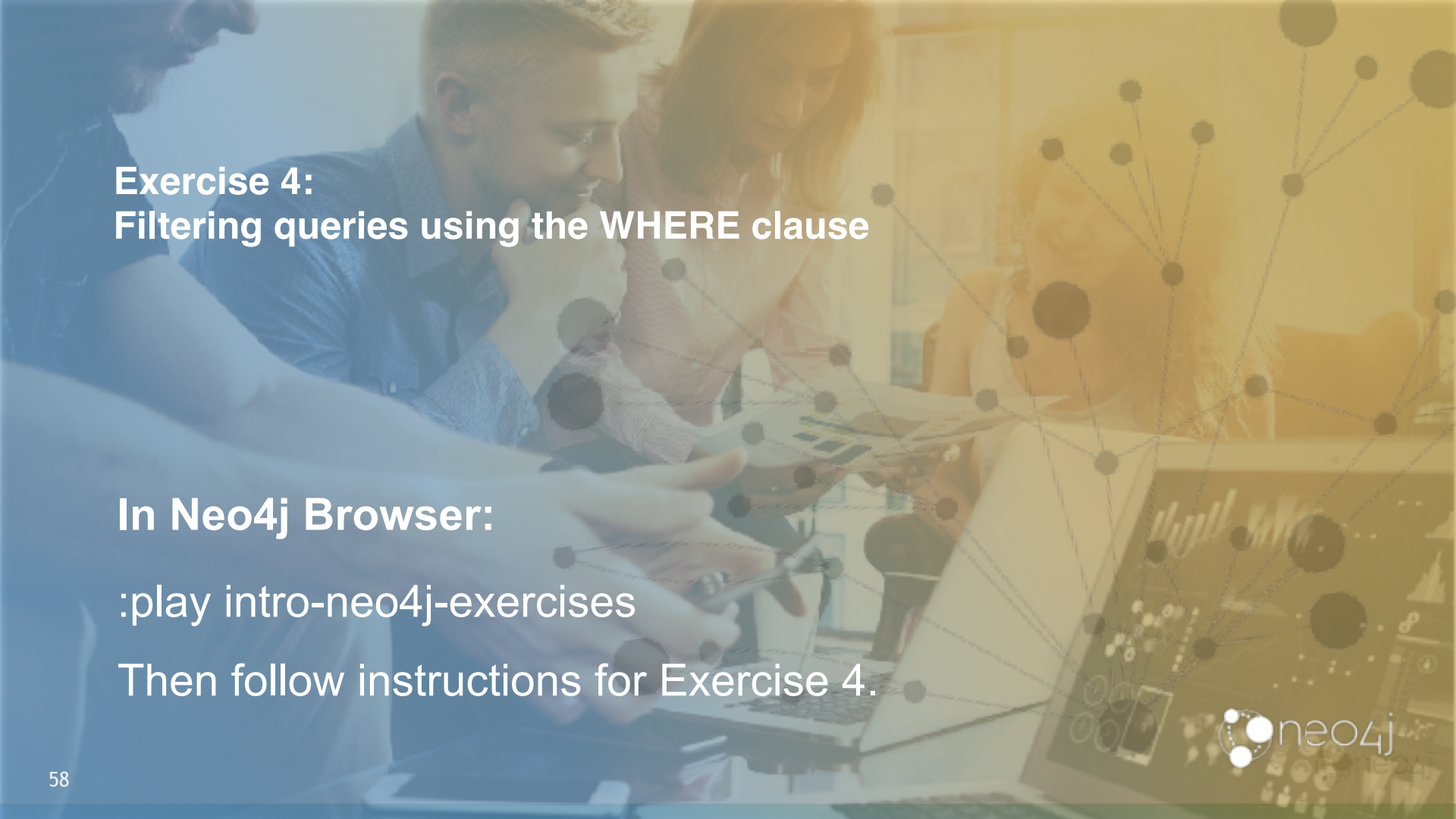
```
MATCH (p:Person)
WHERE p.born IN [1965, 1970]
RETURN p.name as name, p.born as yearBorn
```



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
```



Exercise 4:
Filtering queries using the WHERE clause

In Neo4j Browser:

:play intro-neo4j-exercises

Then follow instructions for Exercise 4.

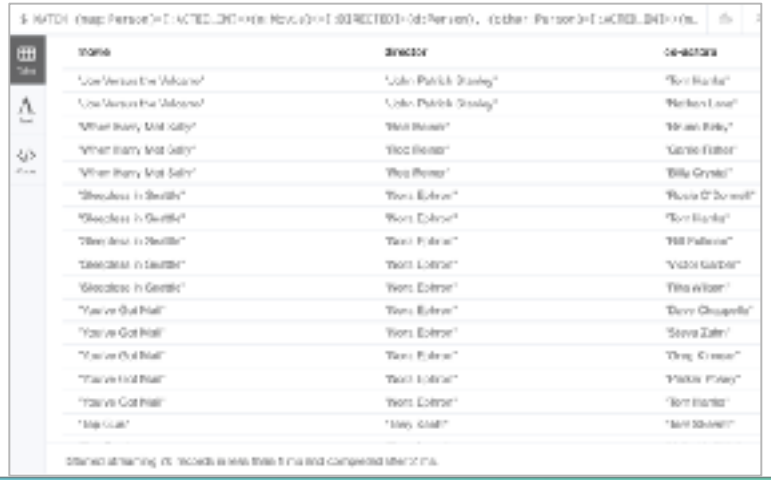
Controlling query processing

- 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:

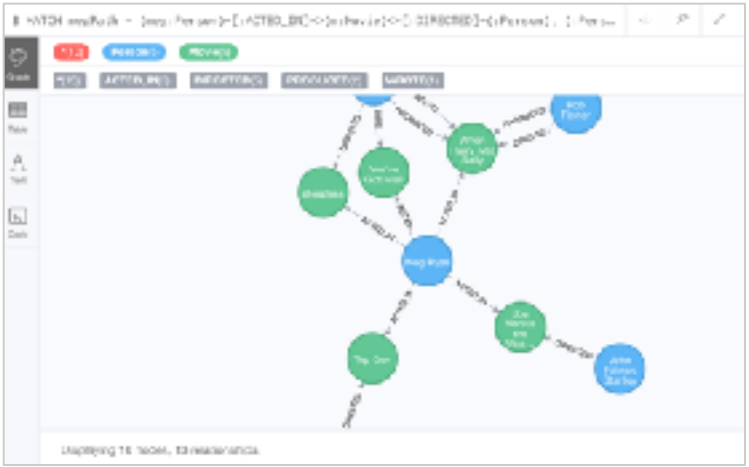
```
MATCH (meg:Person)-[:ACTED_IN]->(m:Movie)<-[:DIRECTED]-(d:Person),
      (other:Person)-[:ACTED_IN]->(m)
WHERE meg.name = 'Meg Ryan'
RETURN m.title as movie, d.name AS director , other.name AS `co-actors`
```



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
```

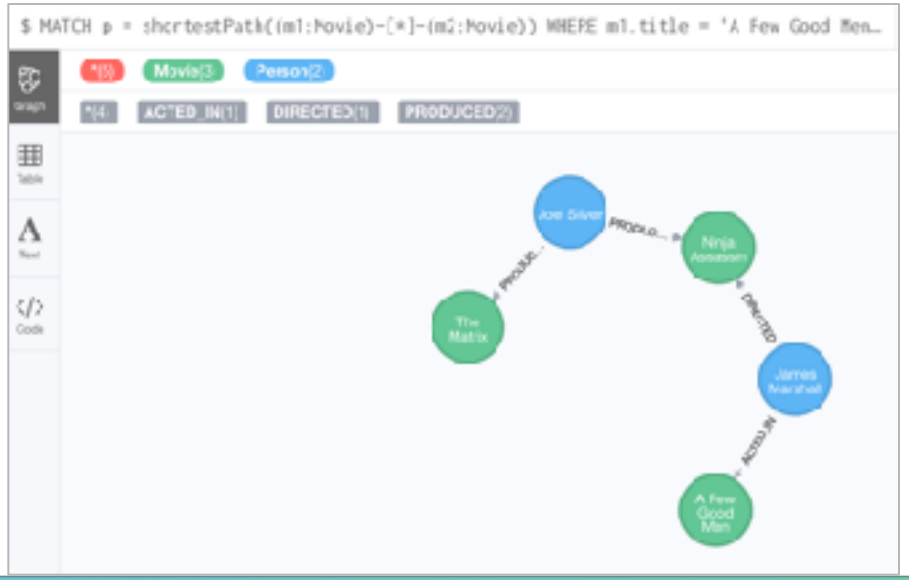


Finding the shortest path

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

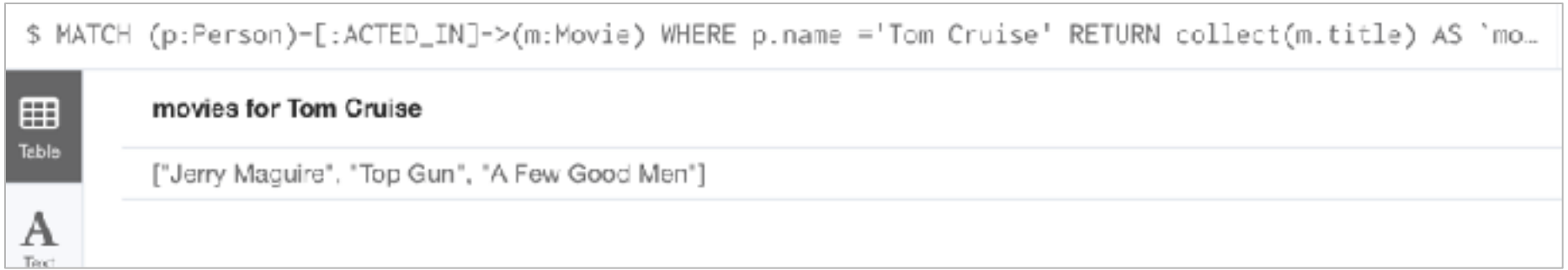
- Different from SQL - no need to specify a grouping key.
- As soon as you use an aggregation function, all non-aggregated result columns automatically become grouping keys.
- Implicit grouping based upon fields in the RETURN clause.

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

Collecting results

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

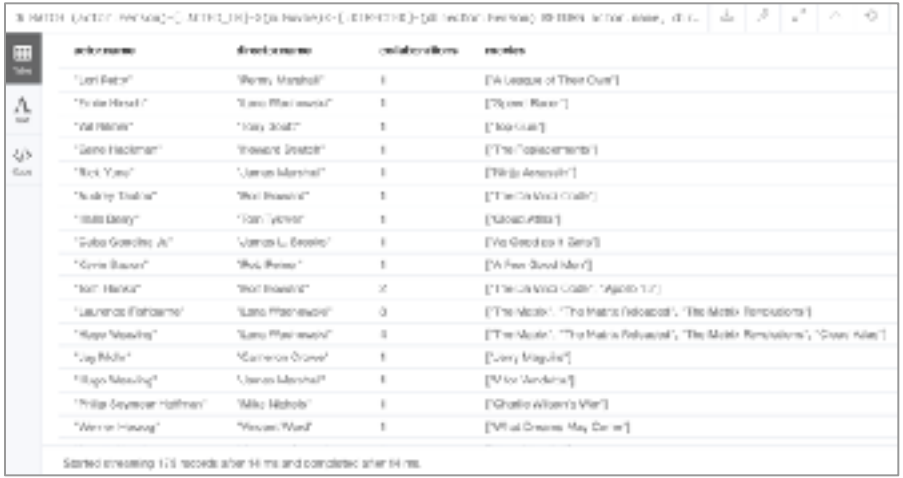
```
MATCH (p:Person)-[:ACTED_IN]->(m:Movie)
WHERE p.name ='Tom Cruise'
RETURN collect(m.title) AS `movies for Tom Cruise`
```



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
```

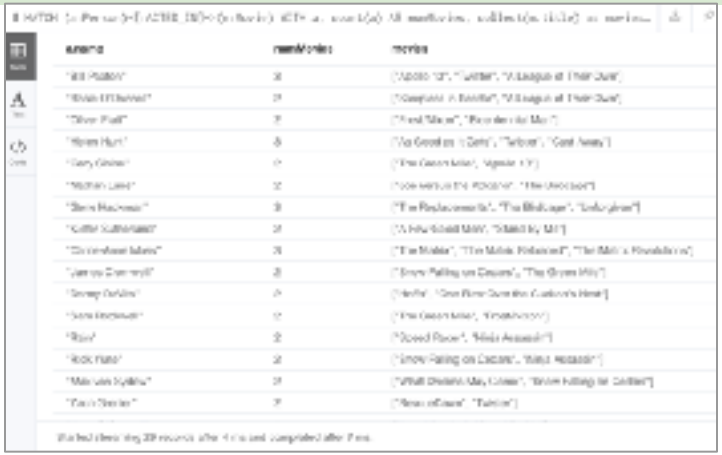


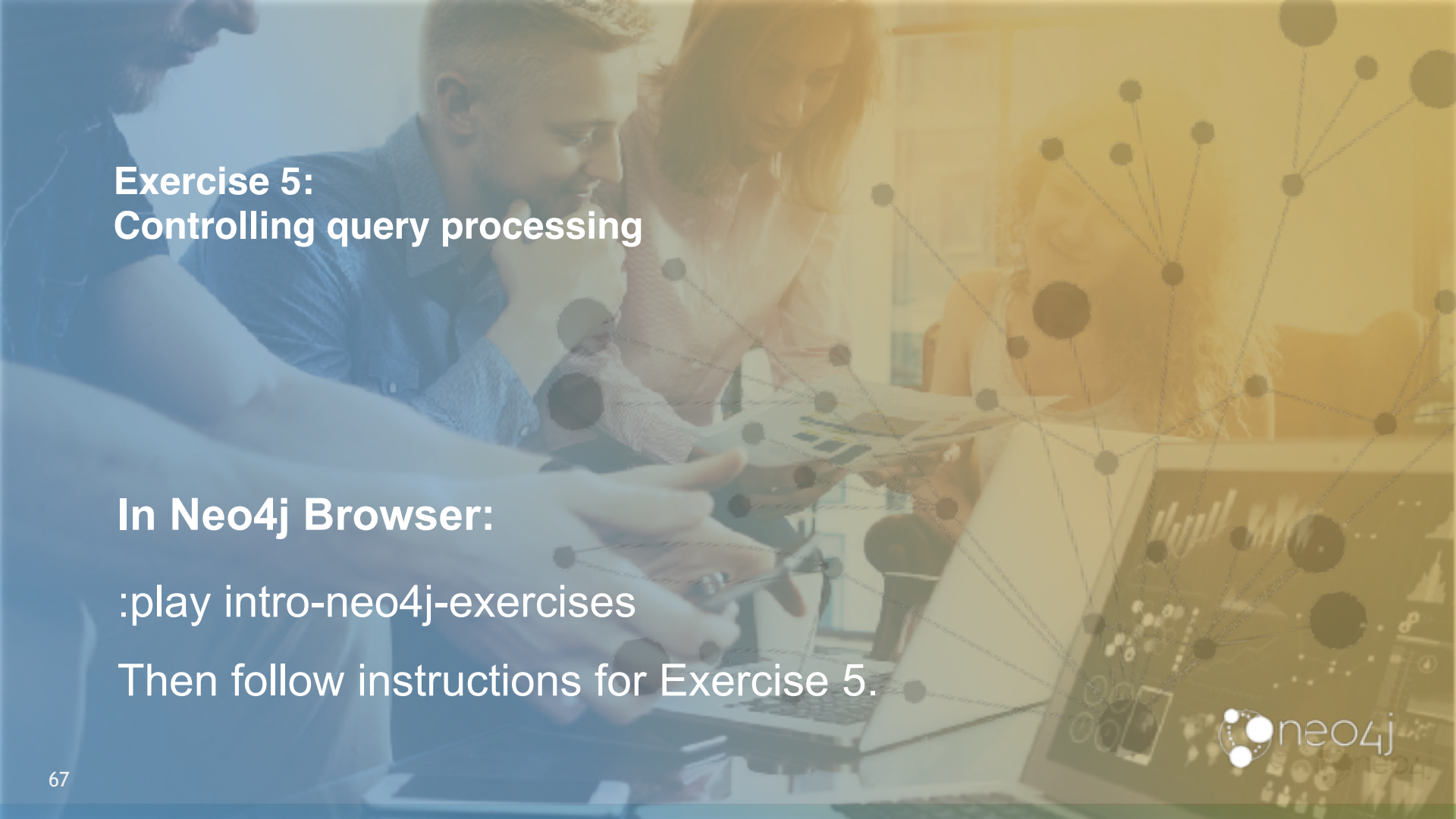
Additional processing using WITH - 1

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
```





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
```

neo4j

Using WITH and DISTINCT to eliminate duplication

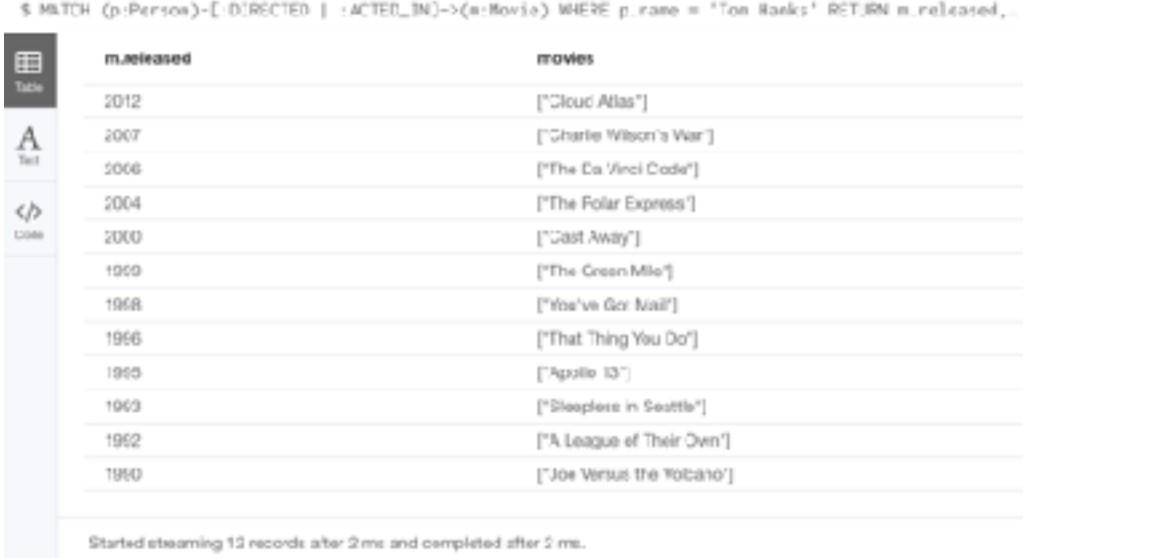
We can also eliminate the duplication in this query by specifying WITH DISTINCT as follows:

```
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

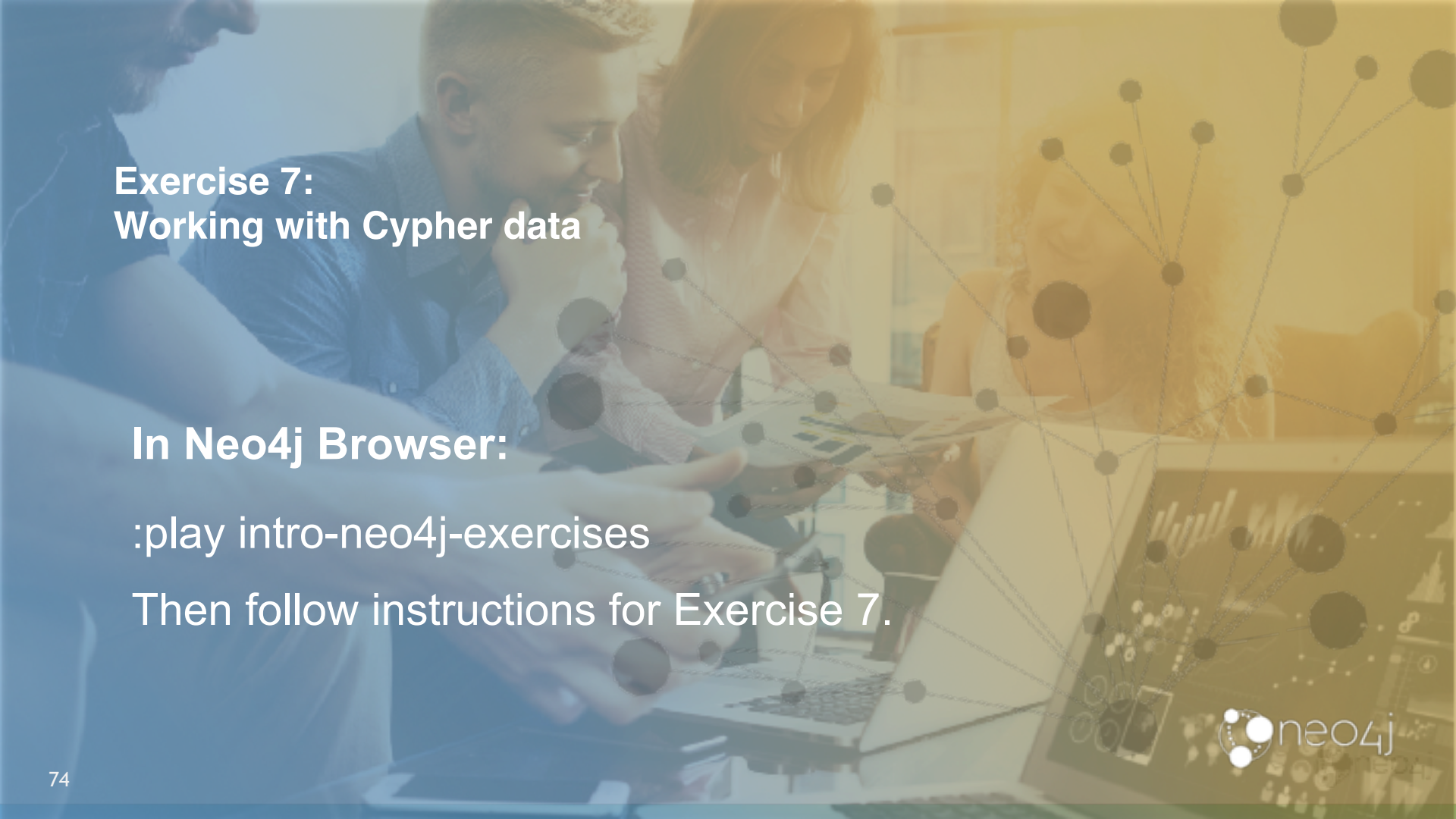
What are the titles of the ten most recently released movies:

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

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]



Exercise 7:
Working with Cypher data

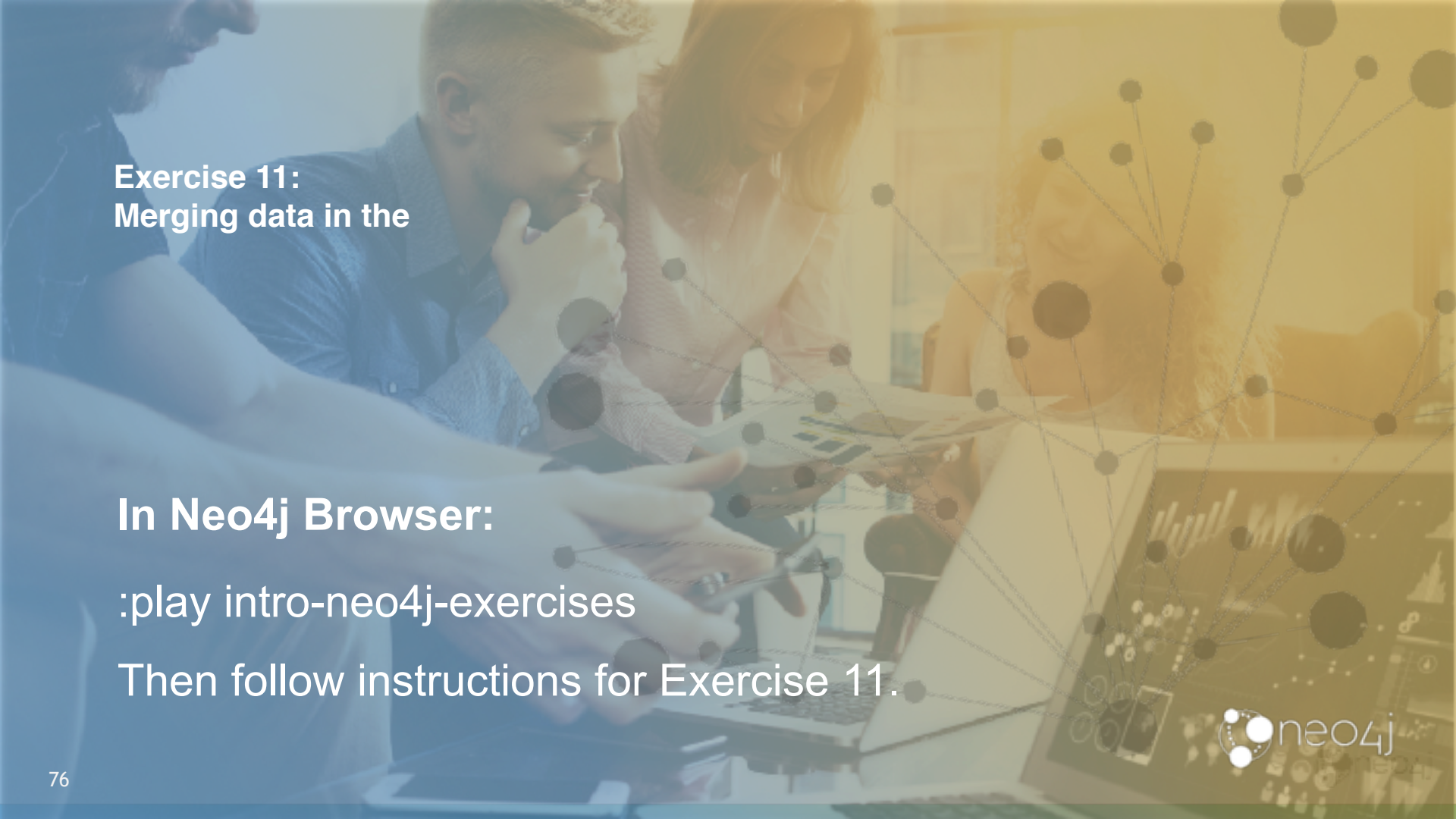
In Neo4j Browser:

: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.



Exercise 11:
Merging data in the

In Neo4j Browser:

:play intro-neo4j-exercises

Then follow instructions for Exercise 11.


Getting More Out of
Neo4j
v 1.0
neo4j

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.
```

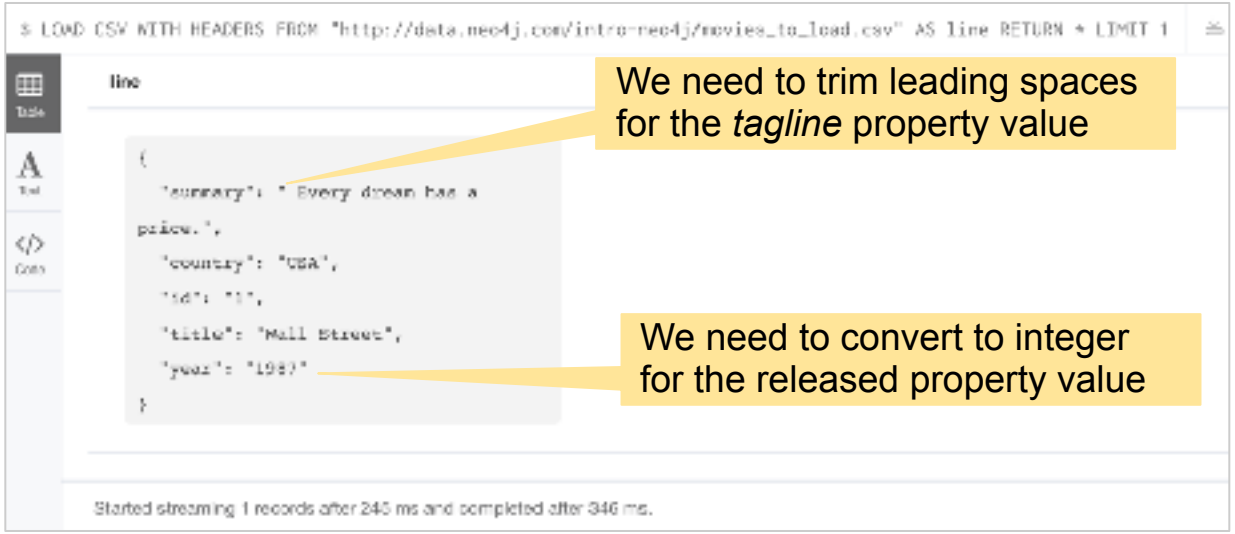
1. Determine the number of lines that will be loaded:

```
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)	Trim(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)})
```

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))
```

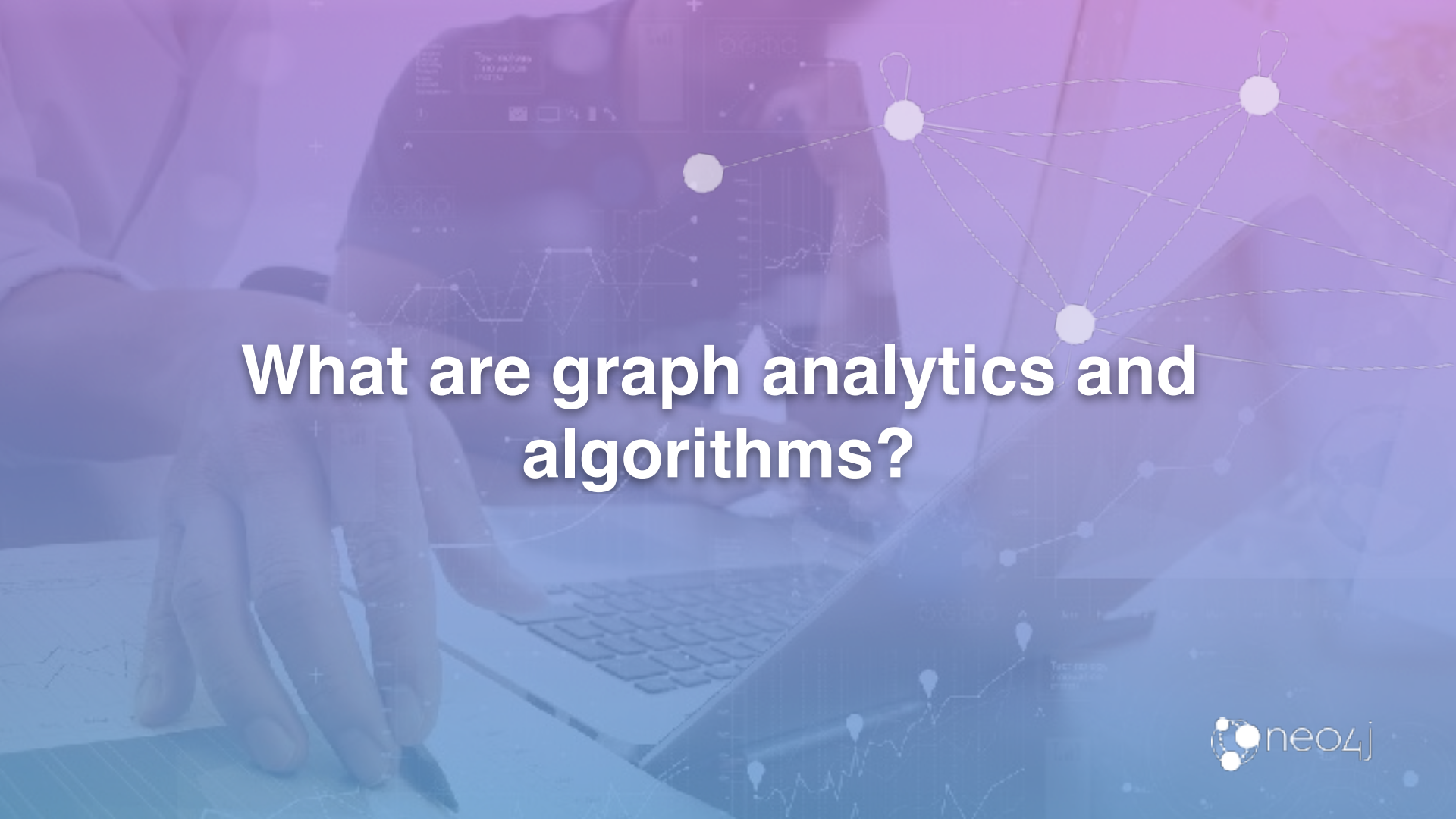
We use MERGE to ensure that we will not create any duplicate nodes

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)
```


What are graph analytics and algorithms?
neo4j

Common types of graph algorithms

Pathfinding & Search

Centrality / Importance

Similarity

Link Prediction

Community Detection+

Basic stats

Decomposition, covering & coloring

Network flow & percolation

Subgraph & isomorphism

ML Workflow

Assortative mixing

Classic graph algorithms categories

Other common categories

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

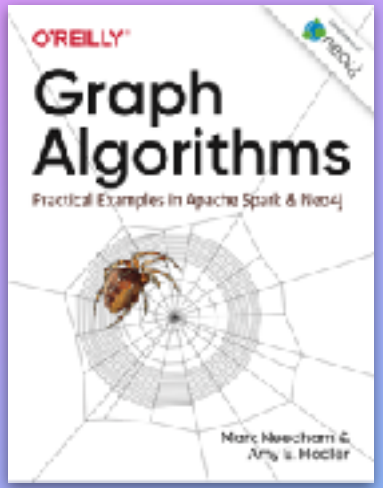
Detects group clustering or partition options.

Estimates the likelihood of nodes forming a future relationship.

Link Prediction

Evaluates how alike nodes are.

Similarity



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

Updated October 2019

neo4j.com/docs/graph-algorithms/current

Centrality algorithms

Determines the importance of distinct nodes in the network.

Developed for distinct uses or types of importance.

Some Centrality algorithms

Degree

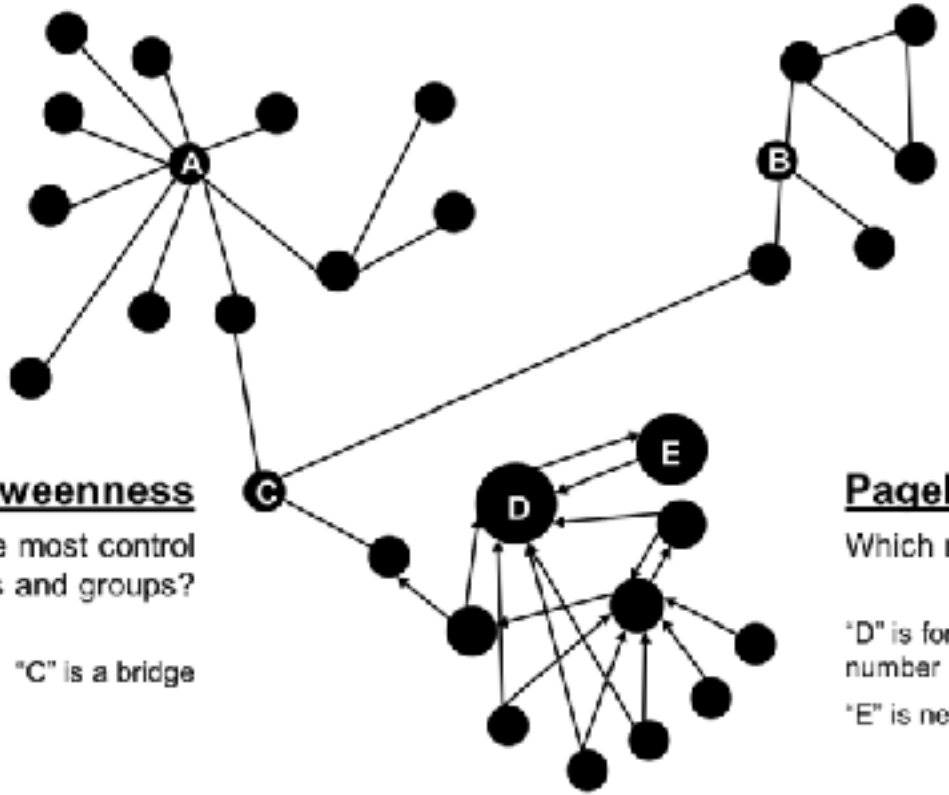
Number of connections?

"A" has a high degree

Closeness

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

"B" is closest with the fewest hops in it's subgraph



Betweenness

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

"C" is a bridge

PageRank

Which node is the most important?

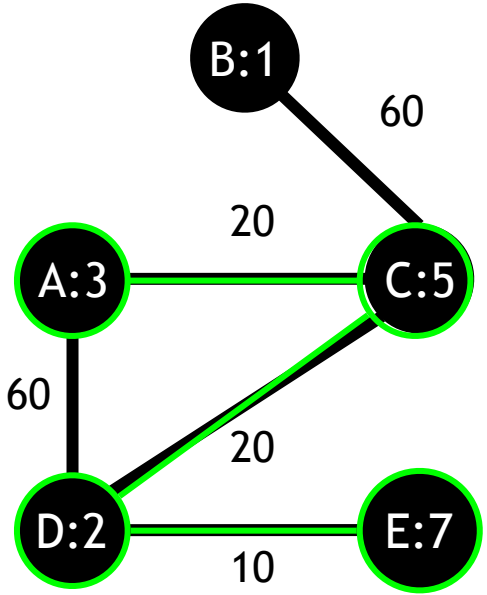
"D" is foremost based on number & weighting of in-links

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

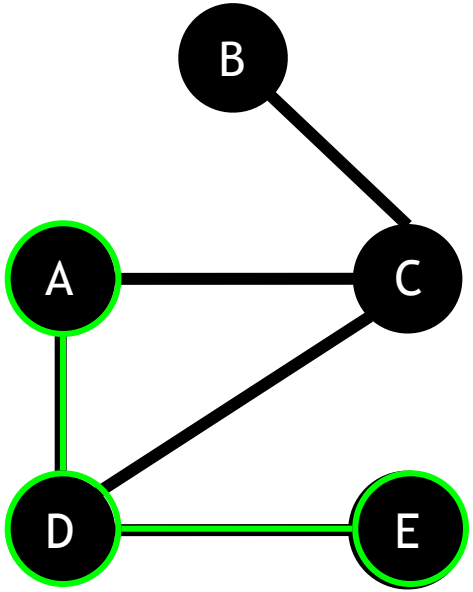
Pathfinding and Graph Search algorithms

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

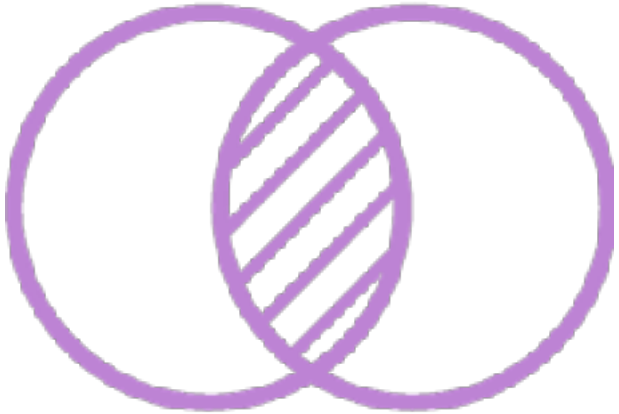


Weighted



Unweighted

Similarity Algorithms

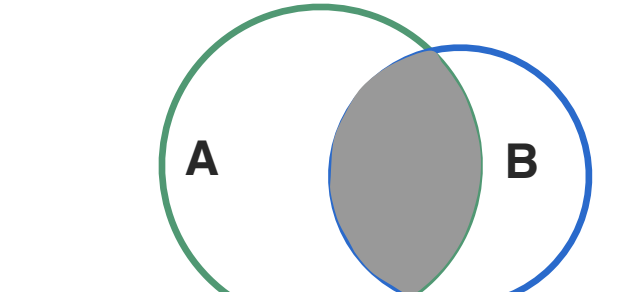


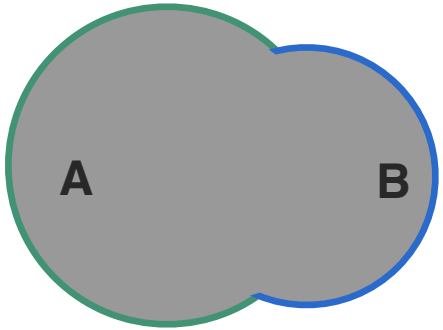
Evaluates how alike nodes are at an individual level either based on node attributes, neighboring nodes, or relationship properties

Jaccard Similarity coefficient

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

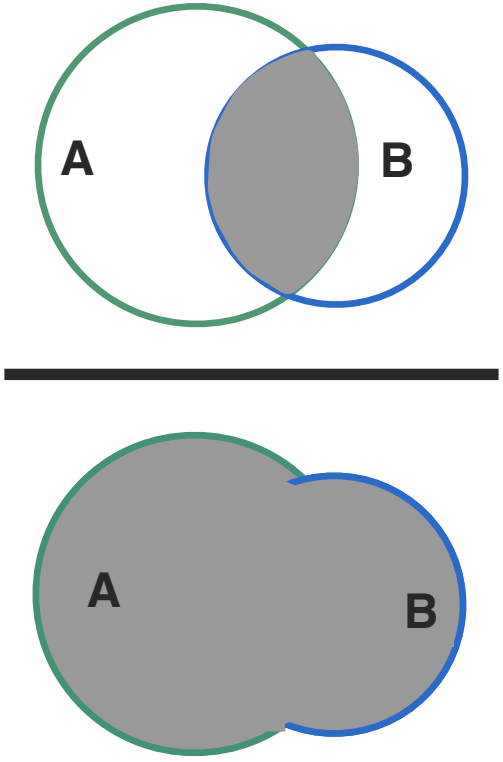
Jaccard Similarity measures the similarity between sets.





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

Jaccard Similarity coefficient



A = {Orange, Banana, Cherry, Pineapple}
B = {Orange, Banana, Apple}

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

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

$|A \cup B|$ = |{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
Field Engineer