# Introduction to Graph and Graph Database

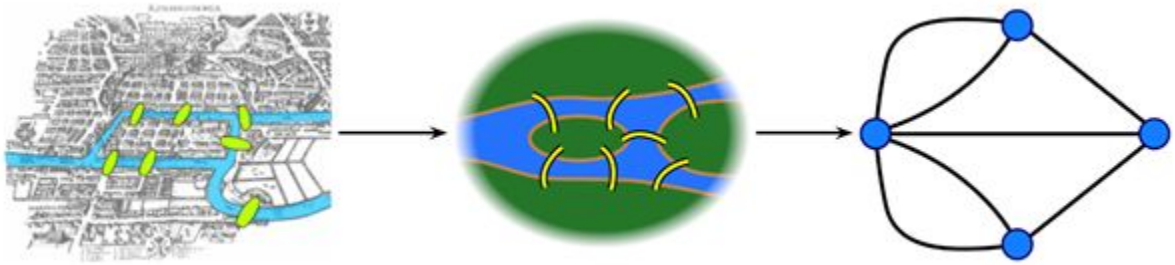
Dr. Sirasit Lochanachit

# Today’s Outline

- Graphs
  - What is a graph?
  - Introduction to property graph model
- Graph Databases
  - What is a graph database?
  - Neo4j

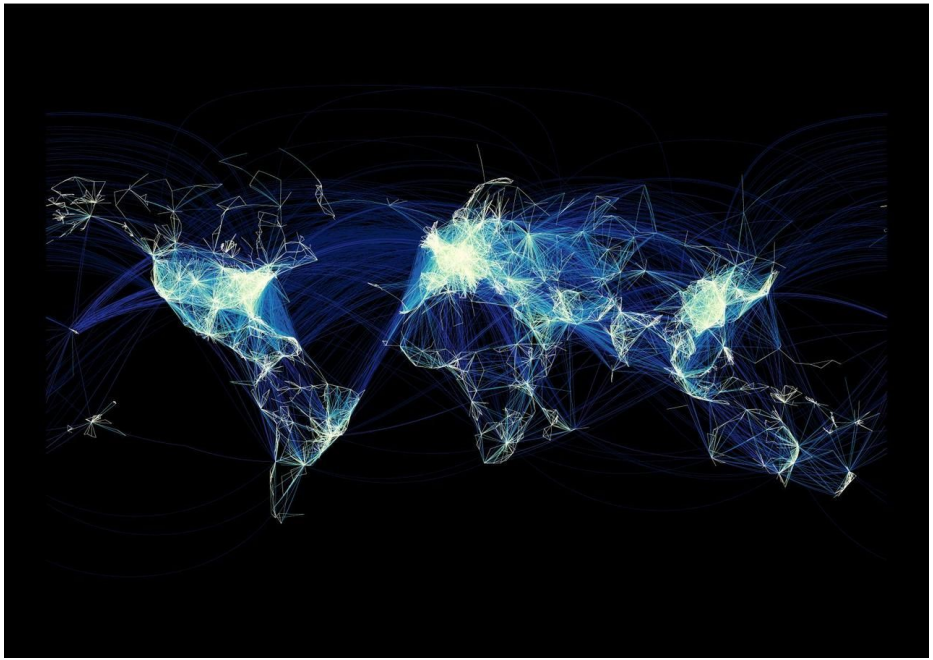
# What is a graph?

- A graph is a set of discrete objects, each of which has some set of relationships with the other objects.



# Anything can be a graph

- Any dataset can be represented as a graph.

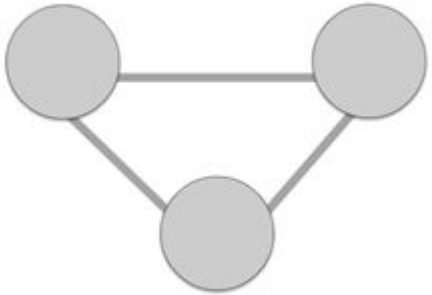


# Everything is connected

- People, places, events
- Countries, history, politics
- Networks, applications, users
- Software, dependencies, architecture, deployments
- etc.

# Property Graph Model

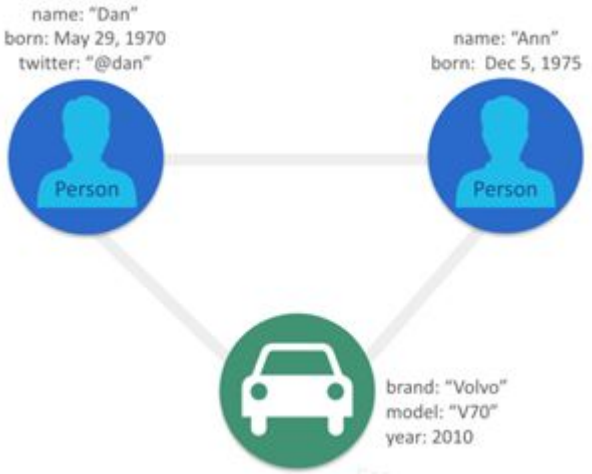
- Nodes
- Relationships
- Properties
- Labels



# Property Graph Model Components

## Node Properties

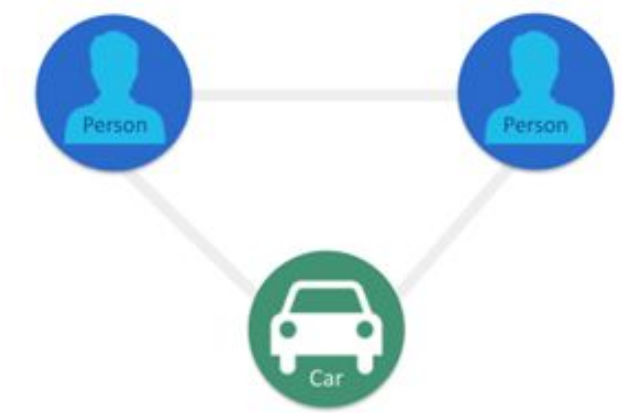
- Name-value pairs that associate specific information with individual nodes



# Property Graph Model Components

**Nodes**

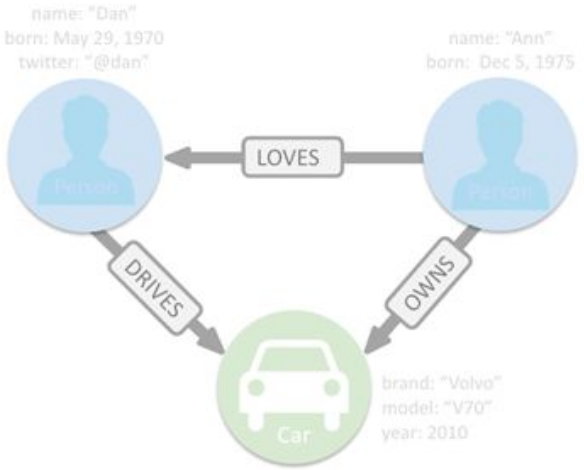
- Represent the objects in the graph
- Can be *labeled*



# Property Graph Model Components

## Relationships

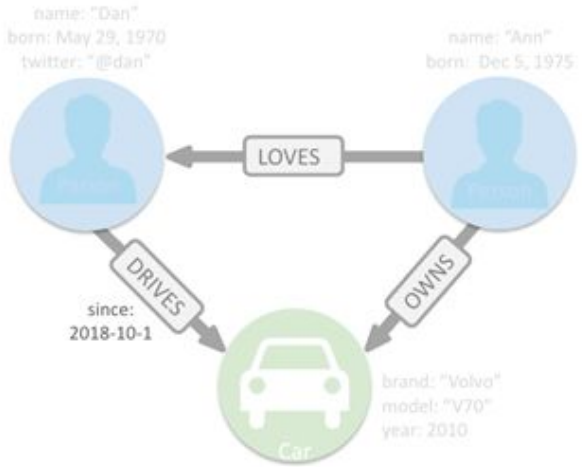
- Relate nodes by *type* and *direction*



# Property Graph Model Components

## Relationship properties

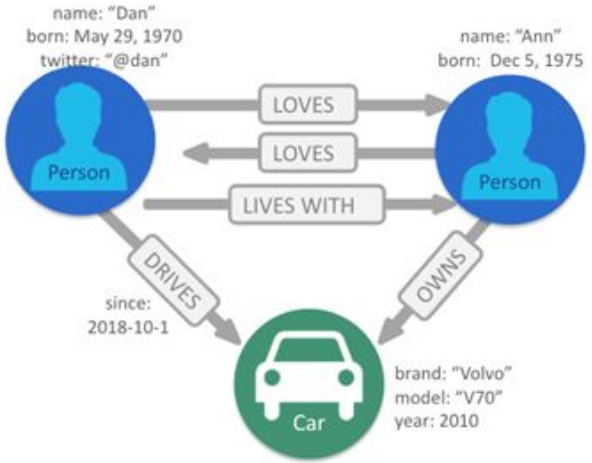
- Express specific attributes



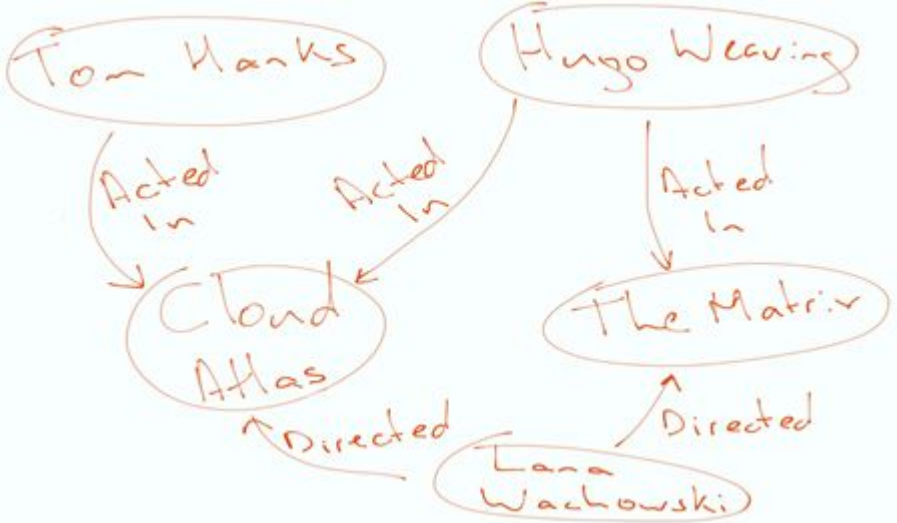
# Property Graph Model Components

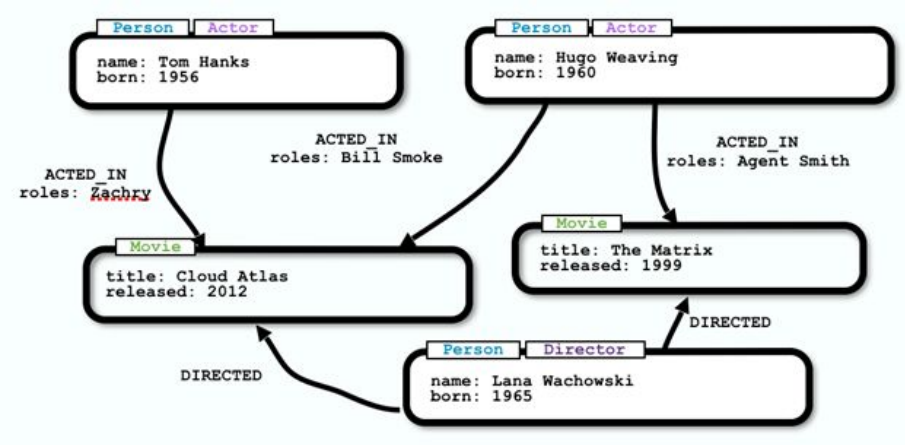
## Multiple relationships

- Each node can have many relationships with other nodes to fully capture their context.



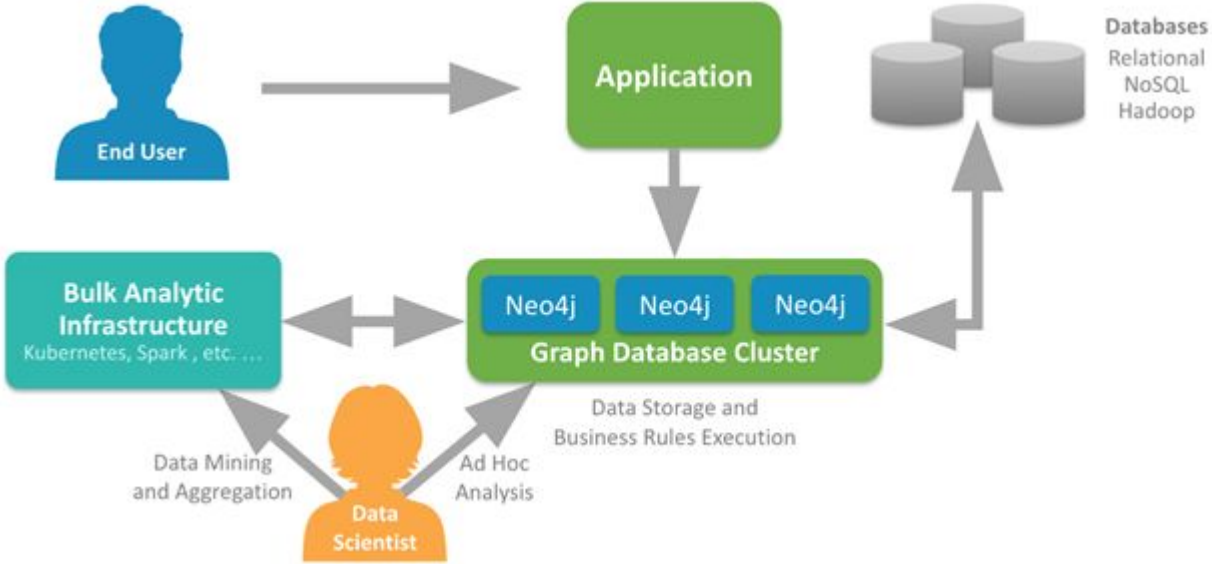
# Whiteboard Friendliness



# What is a graph database?

- Online database management system with Create, Read, Update, and Delete (CRU) operations working on a graph data model.



# Graph DB vs RDBMS

- Graph <> Table
- Node <> Row
- Properties <> Column

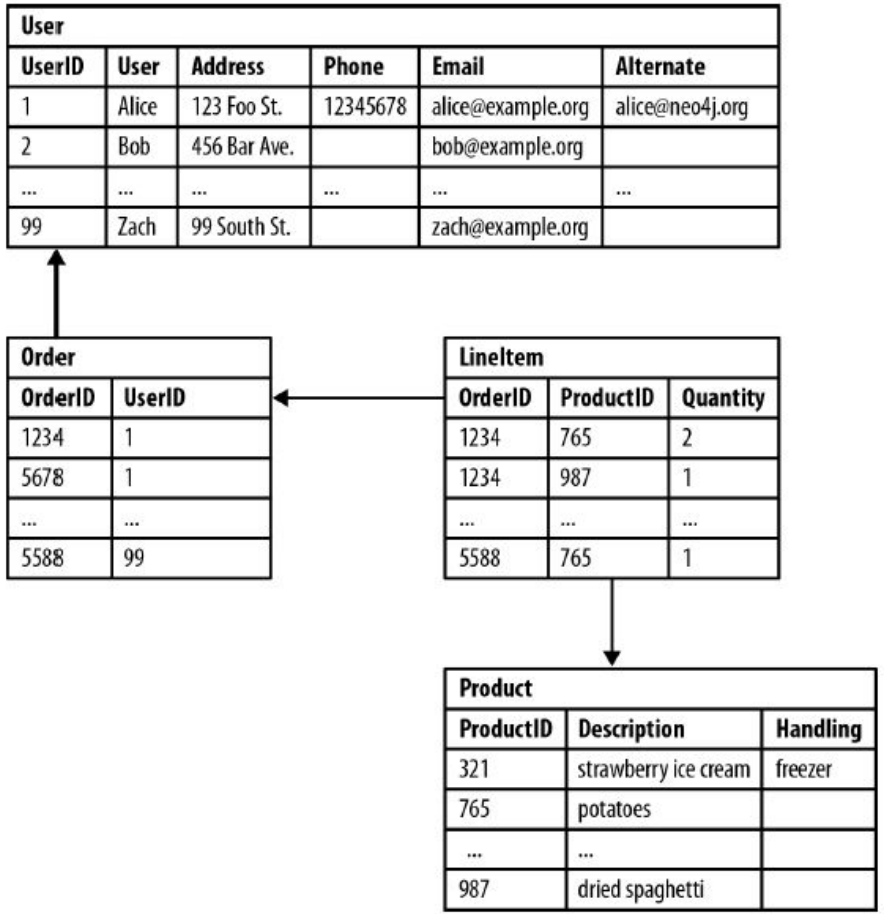
# Power of Graph Databases

- Performance
- Flexibility
- Agility

# Relational Schema

What items did a customer buy?

Which customers bought this product?



**User**

| UserID | User | Address | Phone | Email | Alternate |
|---|---|---|---|---|---|
| 1 | Alice | 123 Foo St. | 12345678 | alice@example.org | alice@neo4j.org |
| 2 | Bob | 456 Bar Ave. | | bob@example.org | |
| ... | ... | ... | ... | ... | ... |
| 99 | Zach | 99 South St. | | zach@example.org | |

**Order**

| OrderID | UserID |
|---|---|
| 1234 | 1 |
| 5678 | 1 |
| ... | ... |
| 5588 | 99 |

**LineItem**

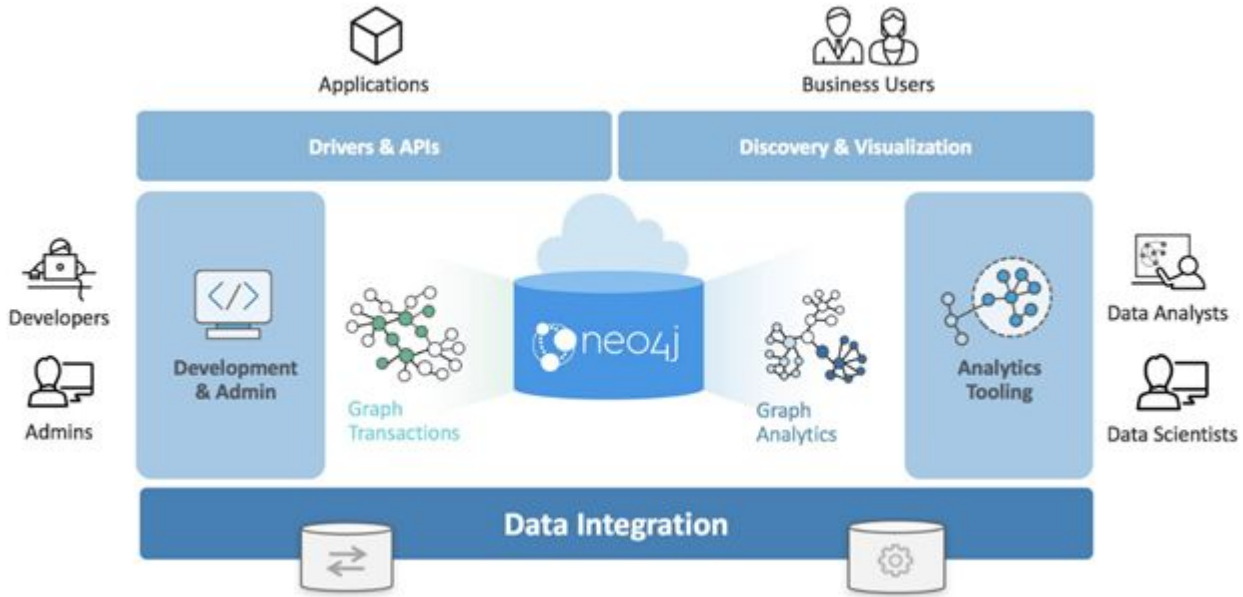
| OrderID | ProductID | Quantity |
|---|---|---|
| 1234 | 765 | 2 |
| 1234 | 987 | 1 |
| ... | ... | ... |
| 5588 | 765 | 1 |

**Product**

| ProductID | Description | Handling |
|---|---|---|
| 321 | strawberry ice cream | freezer |
| 765 | potatoes | |
| ... | ... | |
| 987 | dried spaghetti | |

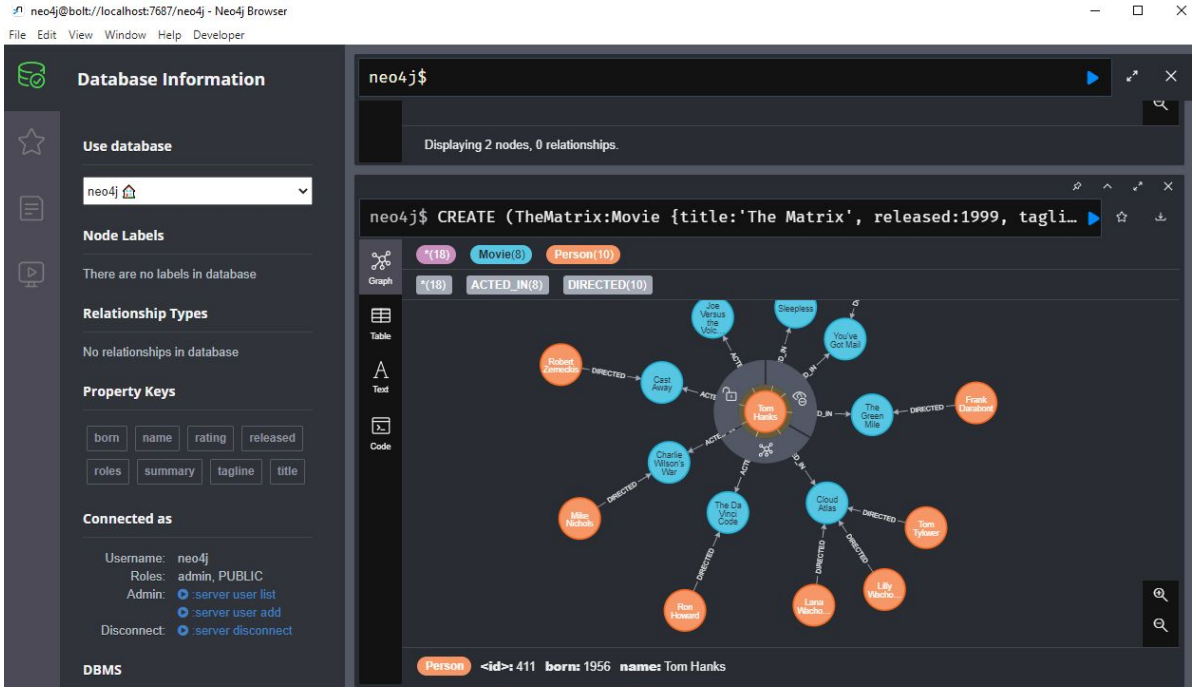
# Neo4j is a graph database

- Neo4j Graph platform is used by developers, administrators, data analysts and data scientists to access application data.



# How to use Neo4j?

1. Create Model
2. Load Data
3. Query
    a. Neo4j Browser



# Who are the Actor/Producers

More Comparisons: https://neo4j.com/graphgists/cypher-vs-sql/

**Sql**

```
SELECT person.name
FROM person
WHERE person.id IN (SELECT person_id FROM acted_in)
  AND person.id IN (SELECT person_id FROM produced)
```

**Cypher**

```
MATCH (person:Person)
WHERE (person)-[:ACTED_IN]->() AND (person)-[:PRODUCED]->()
RETURN person.name
```

# Introduction to Cypher

Dr. Sirasit Lochanachit

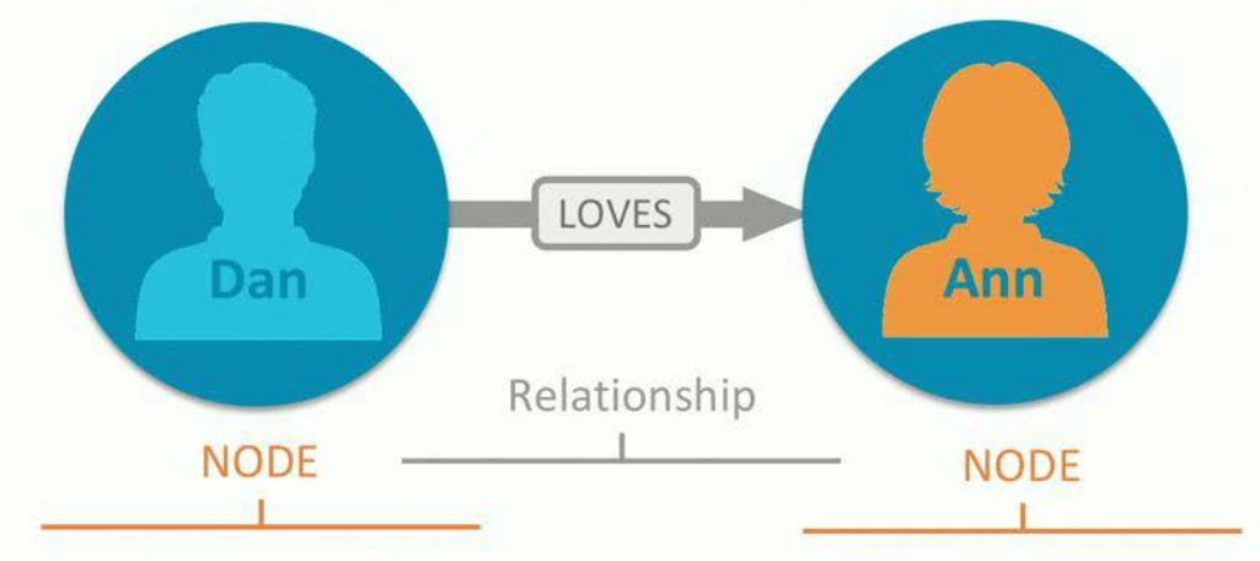
# Today’s Outline

- Filtering Queries
    - Using Nodes, Property Values, and Relationships

# Cypher: The Graph Query Language

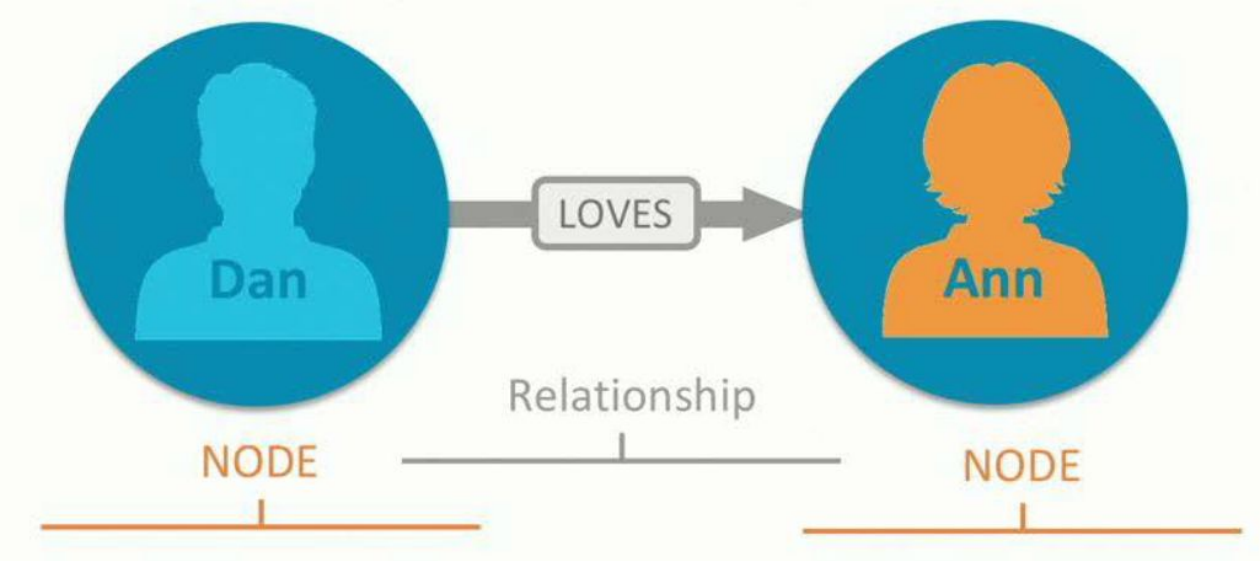
- A pattern matching query language made for graphs
  - Declarative
  - Expressive
  - Pattern matching

# Pattern in Graph Model




Ref: https://www.microsoft.com/en-us/research/uploads/prod/2019/03/41970_Introduction_to_Neo4j_and_Graph_Databases.pdf

# MATCH Graph Patterns

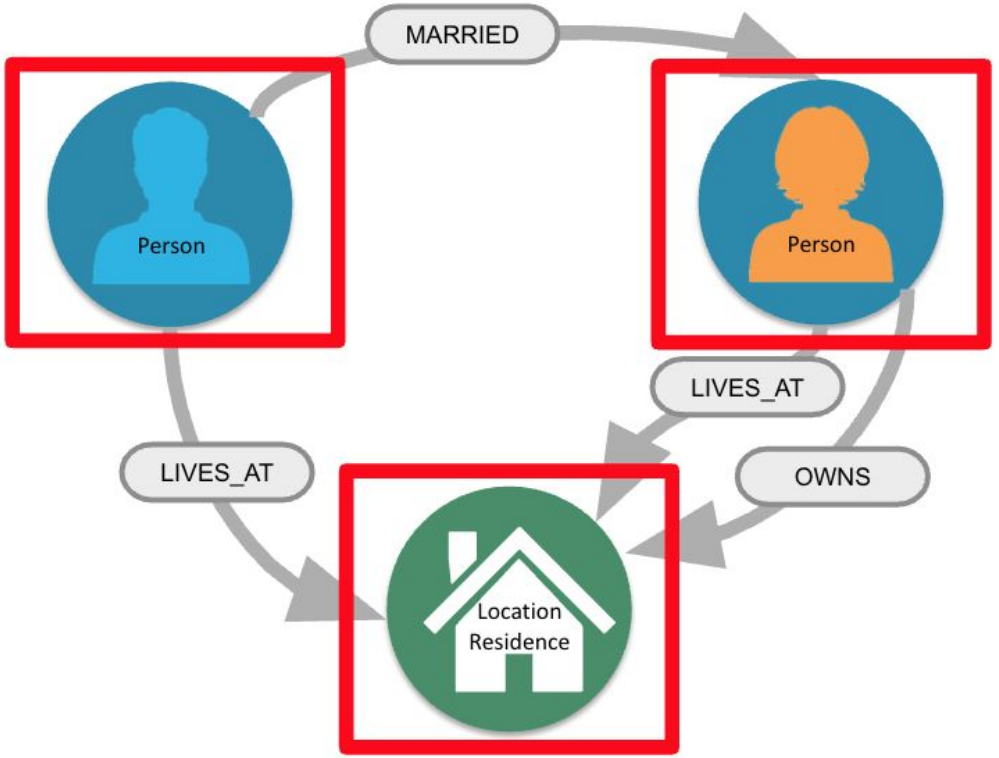



Ref: https://www.microsoft.com/en-us/research/uploads/prod/2019/03/41970_Introduction_to_Neo4j_and_Graph_Databases.pdf

# Node Syntax

()

(<variable>)




Ref: https://neo4j.com/graphacademy/training-querying-40/01-querying40-introduction-to-cypher/

# Label Syntax

(:Person)

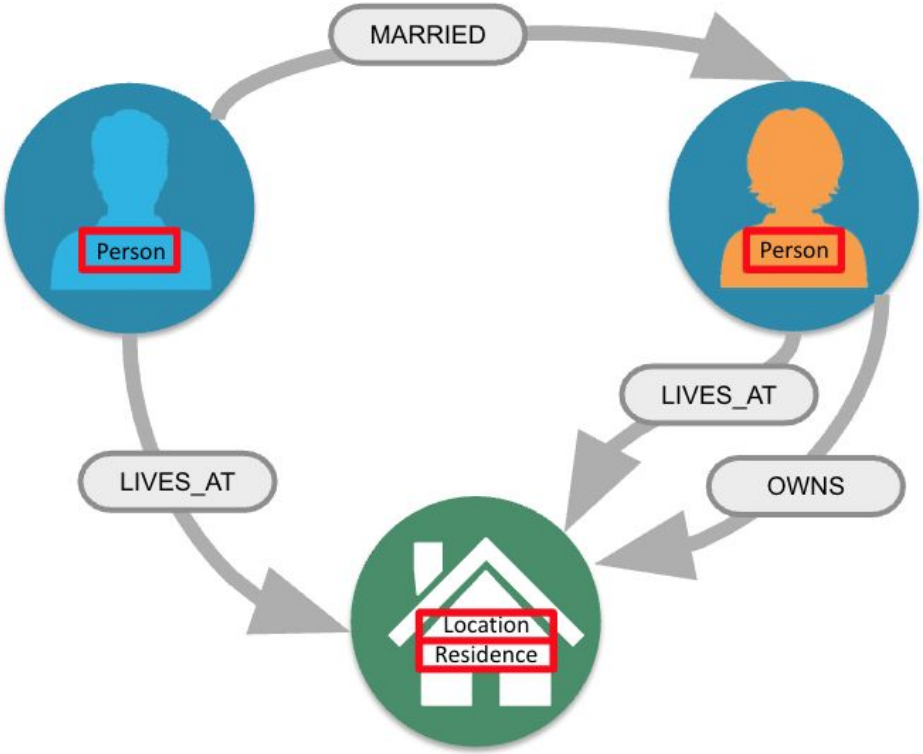
(p:Person)

(:Location)

(l:Location)

(x:Residence)

(x:Location:Residence)



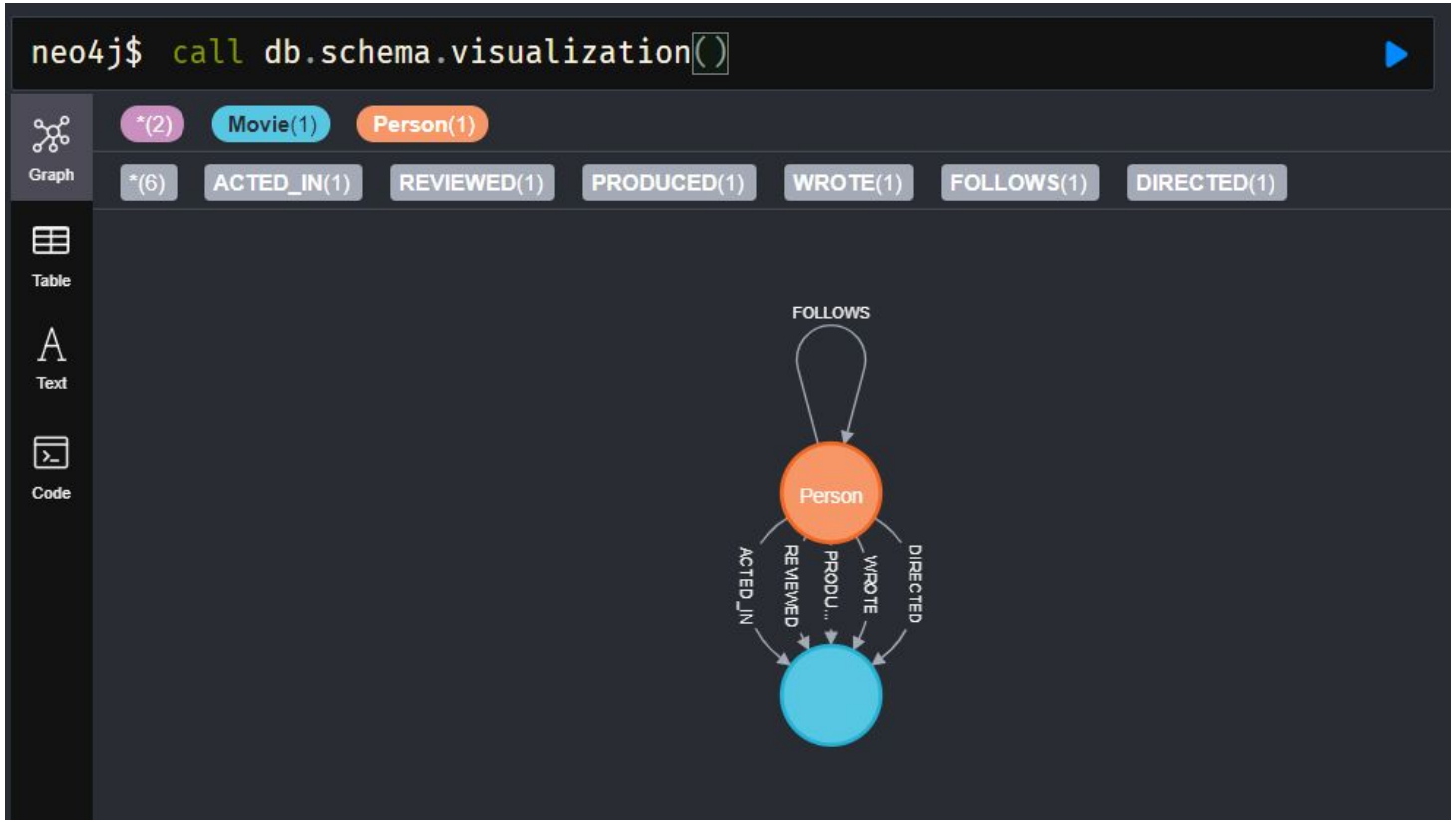

Ref: https://neo4j.com/graphacademy/training-querying-40/01-querying40-introduction-to-cypher/

# Comments

- Place a comment with “//”

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

# Visualising Data Model



# MATCH and RETURN

MATCH (variable:Label)
RETURN variable

Example

1) Retrieve all nodes

```
MATCH (n)
RETURN n
```

2) Retrieve all *Person* nodes

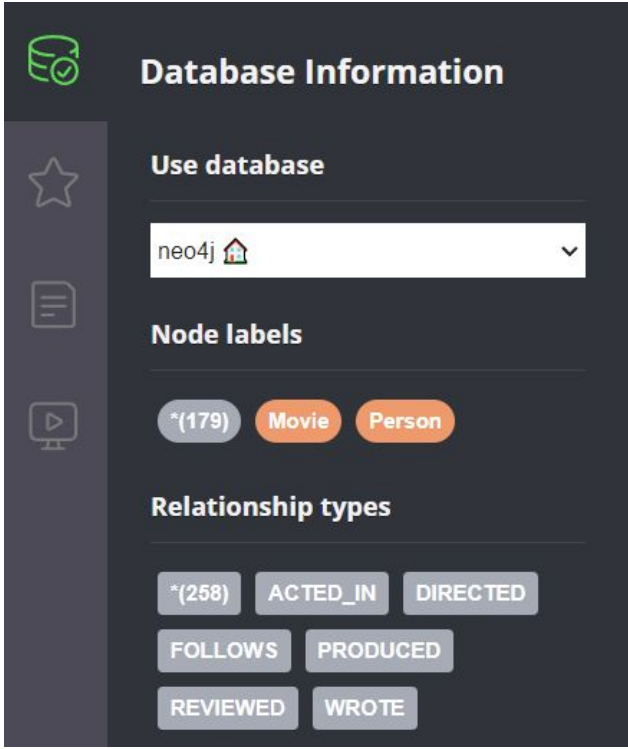
```
MATCH (p:Person)
RETURN p
```

p is a variable

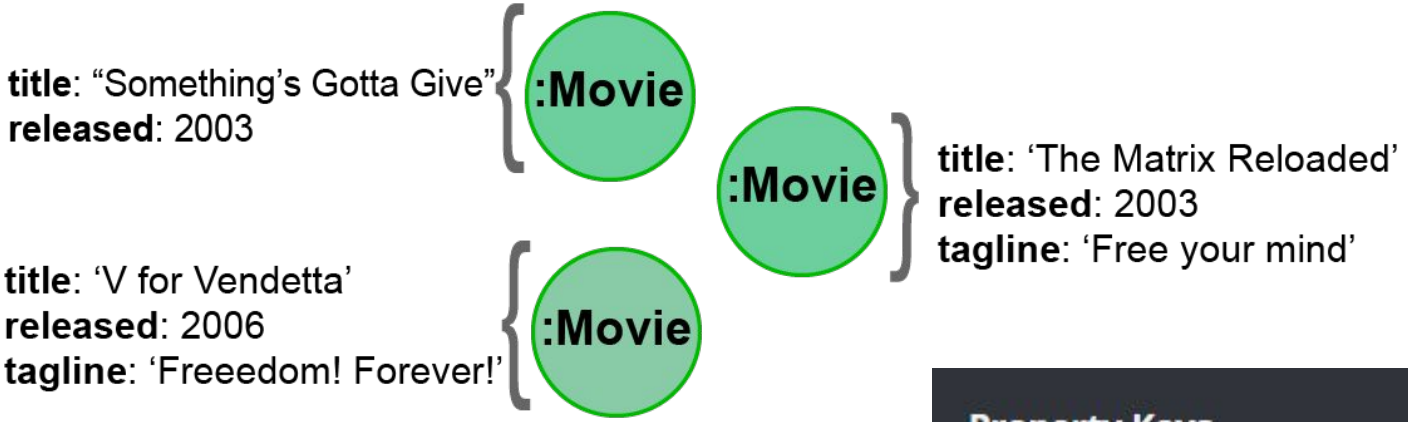
:Person is a node label

# Exercise 1: Retrieving Nodes

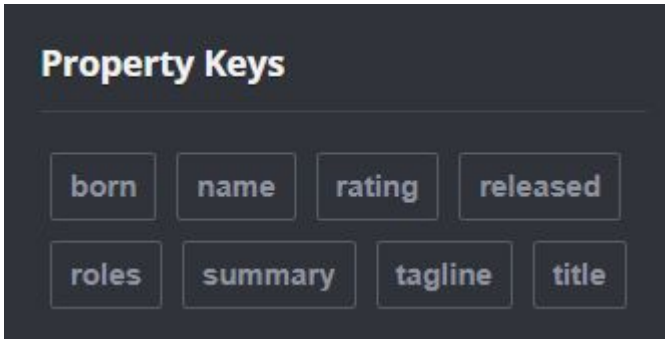
1) Create a new project and create a new DBMS for Movie DB
2) Load and run ‘load-movies.cypher’ and there should be:
    - 179 nodes
    - 258 relationships
3) Write a query to retrieve all *Movie* nodes.



# Properties




CALL db.propertyKeys()




Ref: https://neo4j.com/graphacademy/training-querying-40/01-querying40-introduction-to-cypher/

# MATCH and RETURN

MATCH (variable:Label {propertyKey: propertyValue, propertyKey2: propertyValue2})
RETURN variable

Example

- Retrieve all *Person* nodes that have a *born* property value of *1970*.

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

p is a variable

:Person is a node label

# Returning Property Values

MATCH (variable:Label {prop1: value, prop2: value})
RETURN variable.prop3, variable.prop4

Example

- Retrieve all *name* and *born* values that have a *born* property value of *1970*.

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

p is a variable

:Person is a node label

# Returning Property Values with alias

```
MATCH (variable:Label {prop1: value, prop2: value})
RETURN variable.prop3 AS alias3
```

Example

- Retrieve all *name* and *born* values that have a *born* property value of *1970*.

```
MATCH (p:Person {born: 1970})
RETURN p.name AS name, p.born AS `birth year`
```

# Exercise 2: Filtering Queries using Property Values

1) Write a query to retrieve all *Movie* nodes that *released* in 2019.
2) Write a query to retrieve all Movie nodes that were released in 2020.
3) Write a query to retrieve all *Movie* released in 2003, returning their titles.
4) Write a query to retrieve all *Movie* nodes and display the *title*, *released*, and *tagline* values.
5) Modify 4) query to rename the columns as ‘movie title’, ‘released year’, and ‘tagLine’

# Relationship Syntax

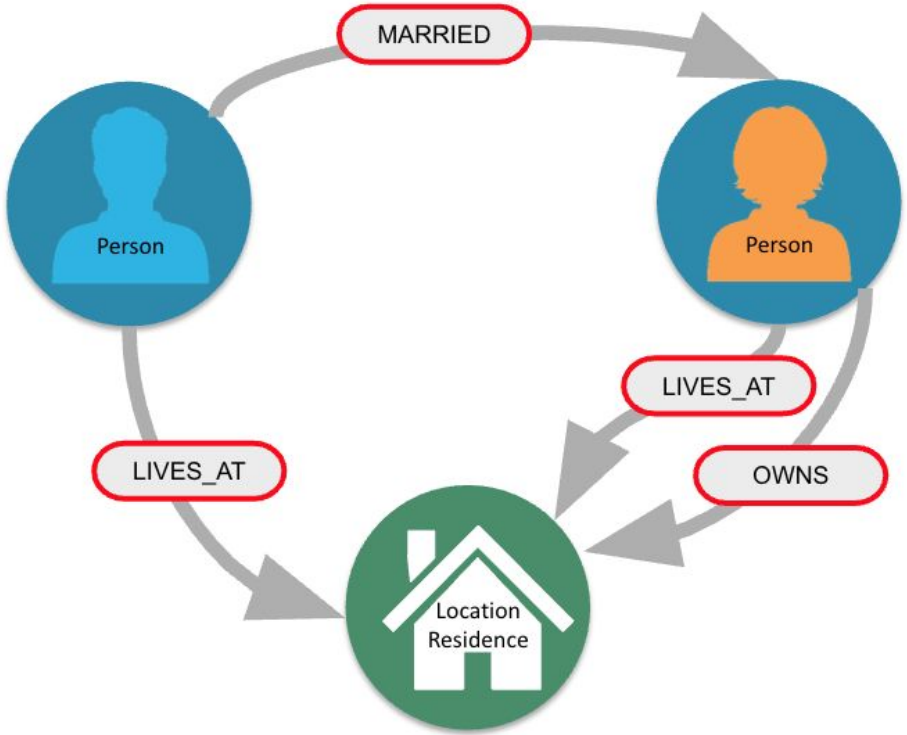
()

() -- ()

() - [] - ()

() --> ()

() <-- ()




Ref: https://neo4j.com/graphacademy/training-querying-40/01-querying40-introduction-to-cypher/

# MATCH and RETURN using relationships

```
MATCH (node1) -[:REL_TYPE_A | REL_TYPE_B]-> (node2)
RETURN node1, node2
```

Example

- Retrieve all *Person* nodes that have acted in *The Matrix*.

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

p, rel, m is a variable

:Person, :Movie is a node label

:ACTED_IN is a relationship label

# Using an Anonymous Relationship for a Query

- Retrieve all *Person* nodes that have any relationship in *The Matrix*.

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

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

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

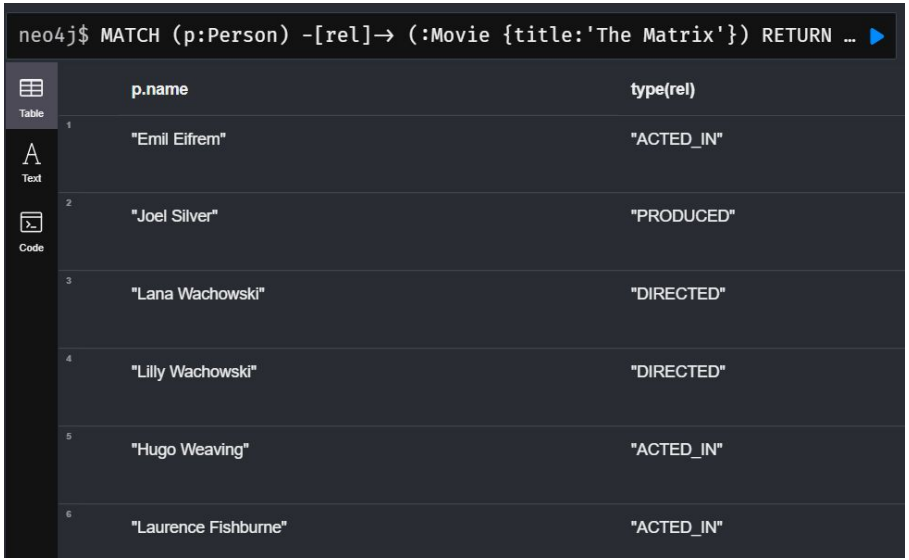
- Retrieve all *Movie* nodes that have any relationship with *Keanu Reeves*.

```
MATCH (m:Movie) <-- (p:Person {name: 'Keanu Reeves'})
RETURN p, m
```

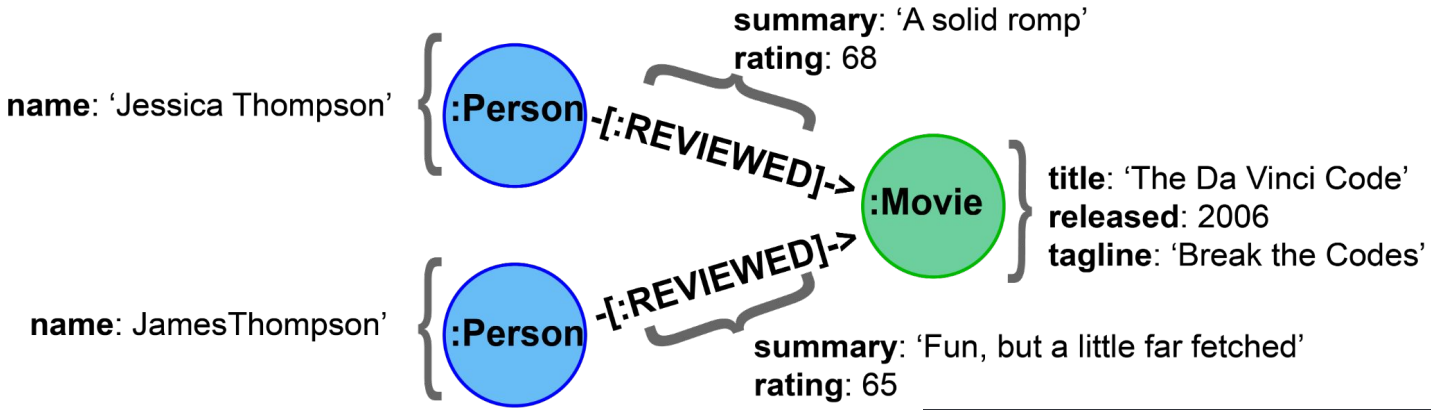
# Retrieving the relationship types

- Retrieve all *Person* nodes that have any relationship in *The Matrix*.

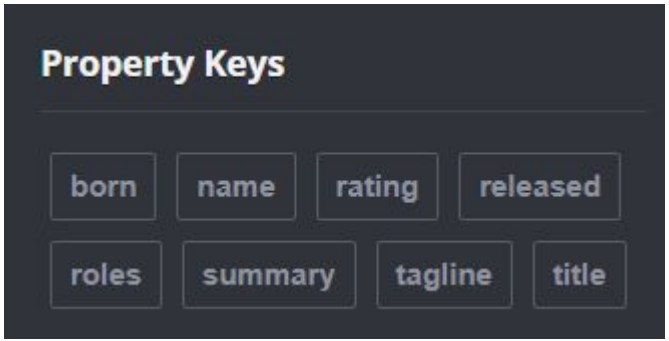
```
MATCH (p:Person) -[rel]-> (:Movie {title:'The Matrix'})
RETURN p.name, type(rel)
```



# Properties for relationships




CALL db.propertyKeys()



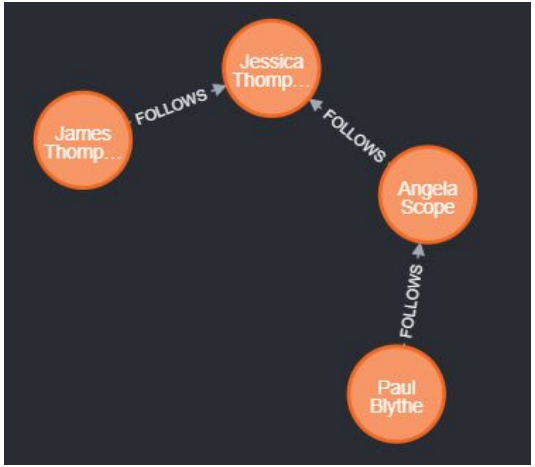

Ref: https://neo4j.com/graphacademy/training-querying-40/01-querying40-introduction-to-cypher/

# MATCH and RETURN using relationship properties

- Retrieve the name of the person who gave *The Da Vinci Code* movie a rating of *65*.

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

# Patterns in the graph



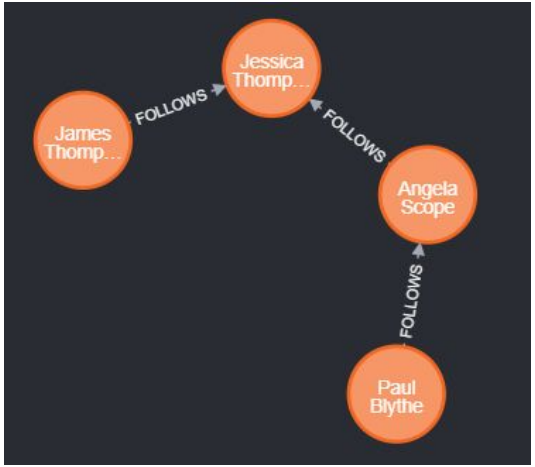

- Retrieve all *Person* nodes who follow *Angela Scope*.

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

- Retrieve all *Person* nodes who is followed by *Angela Scope*.

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

# Patterns in the graph




- Retrieve all *Person* nodes who follow or is followed by *Angela Scope*.

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

# Traversing Multiple Relationships

- Return all followers of the followers of *Jessica Thompson*.

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

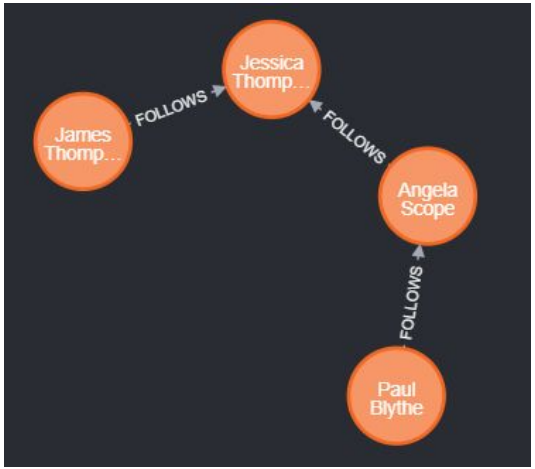
- Return each person name

```
MATCH  (p:Person) -[:FOLLOWS]-> (p2:Person) -[:FOLLOWS]-> (p3:Person {name:'Jessica Thompson'})
RETURN p.name, p2.name, p3.name
```

```
neo4j$ MATCH (p:Person)-[:FOLLOWS]→(p2:Person)-[:FOLLOWS]→(p3:Person …
```

Table | Text

| | p.name | p2.name | p3.name |
|---|---|---|---|
| 1 | "Paul Blythe" | "Angela Scope" | "Jessica Thompson" |



# Exercise 3: Filtering Queries using Relationships

1) Write a query to retrieve all *Person* names who wrote the movie *Top Gun*.
2) Write a query to retrieve all movie titles connected with *Tom Hanks*.

   Hint: Tom Hanks has multiple relationships with a movie (Actor and Director)

3) Modify 2) query to return the information as a table about the type of relationships between Tom Hanks and the movies.
4) Retrieve information about the movies and roles that Tom Hanks acted in.