

Introduction to Neo4j - a hands-on crash course



Michael Hunger
Developer Relations
 @mesirii

dev.neo4j.com/forum
dev.neo4j.com/chat

This is an *interactive* session!
Tell us where you're *from*!

And please ask *Questions*
as we go along!

In any of the chats

In this session



We will cover:

- What is a graph and why they are amazing
- Spotting good graph scenarios
- Property graph database anatomy and introduction to Cypher
- Hands-on: the movie graph on Neo4j AuraDB Free
 - dev.neo4j.com/aura-login
- Sneak peek: stackoverflow data
- Continuing your graph journey

Useful reference: <https://dev.neo4j.com/rdbms-gdb>

Because it takes a few minutes to launch

Let's get started on
AuraDB Free

dev.neo4j.com/aura-login

Ask in the chat if something doesn't work!

Time to have a go! With AuraDB Free



We are going to:

1. Go to dev.neo4j.com/aura-login
2. Sign in & click “Create a database”
3. Selected “AuraDB Free” database size
4. Give your database a name
5. Select a Region
- 6. Select Movies Database**
7. Click “Create Database”
- 8. Make a copy of the generated password - keep it safe!**
9. Wait for 3-5 minutes

Can't access Aura Free? No problem! Use Neo4j Sandbox:

- Go to dev.neo4j.com/try
- Sign in & click “Blank sandbox”

The screenshot shows the Neo4j AuraDB console interface. At the top, there's a navigation bar with icons for window control (red, yellow, green), a refresh, and a search. The URL in the address bar is `console.neo4j.io/#databases`. Below the address bar is a dark header bar with the Neo4j logo, "AURA DB", and links for "DATABASES", "IMPORT", "CONNECT", "FEEDBACK", "GET HELP", and a user profile icon labeled "NE". The main content area features a central illustration of a woman sitting at a desk with a laptop, surrounded by a circular network of nodes. Below the illustration, the text "Let's get started!" is displayed, followed by a description: "Create a new fully-managed Neo4j database in just a few clicks." A blue button labeled "Create a database" is centered below this text.

Let's get started!

Create a new fully-managed Neo4j database in just a few clicks.

[Create a database](#)

console.neo4j.io/#create-database-wiz

Let's create your first database

Step 1 of 1

Database type

AuraDB Free AuraDB Professional

more detail ▾

Database details

Database Name: Movies

GCP Region: Iowa, USA (us-central1)

Starting dataset

 Learn about graphs with a movie dataset
Beginner

 Load or create your own data in a blank database

Cancel Create Database

Credentials for Movies

Username

neo4j 

Generated password

yHK6r8BAUd2uGe5KQn7H7kSU3pcp7XelumLrduiG1L4 

Once dismissed, these credentials will no longer be retrievable. We recommend you update the password once your database is created.

I have stored these credentials safely to use later

Continue

console.neo4j.io/#databases/fa05846a/de

neo4j QUORADB DATABASES IMPORT CONNECT FEEDBACK GET HELP NE

Movies **FREE**

Running

Connection URI: neo4j+s://fa05846a.databases.neo4j.io

Region: Iowa, USA (us-central1)

Neo4j Version: 4

Nodes: 0 / 50000 (0%)

Relationships: 0 / 175000 (0%)

Open with

- Neo4j Browser
- Neo4j Bloom (beta)

IMPORT DATABASE CONNECT SNAPSHOT SETTINGS

Python Javascript GraphQL Java Spring Boot .NET Go

```
1 from neo4j import GraphDatabase
2 import logging
3 from neo4j.exceptions import ServiceUnavailable
4
5 class App:
6
7     def __init__(self, uri, user, password):
8         self.driver = GraphDatabase.driver(uri, auth=(user, password))
9
10    def close(self):
11        # Don't forget to close the driver connection
12        self.driver.close()
13
14    def create_friendship(self, person1_name, person2_name):
15        with self.driver.session() as session:
16            # Write transactions allow the driver to handle errors
17            result = session.write_transaction(
18                self._create_and_return_friendship,
19                person1_name, person2_name)
20
21            for row in result:
22                print(row)
```

Prerequisites

- Python 3.4+

Downloading and Installing the Neo4j Driver

Install the Neo4j Driver using pip:

```
pip install neo4j
```

Neo4j Aura

← Neo4j Browser Guides

aura/movies

What is Cypher?

Cypher is a graph query language that is used to query the Neo4j Database. Just like you use SQL to query a MySQL database, you would use Cypher to query the Neo4j Database.

A simple cypher query can look something like this

```
④ Match (m:Movie) where m.released > 2000 RETURN m limit 5
```

Hint: You can click on the query above to populate it in the editor.

Expected Result: The above query will return all the movies that were released after the year 2000 limiting the result to 5 items.

Cloud Atlas
RescueDawn
The Matrix Revolutions
The Da Vinci Code

1 2 3 4 5 ... 11 Next

Neo4j Browser

\$

Database access not available. Please use `:server connect` to establish connection. There's a graph waiting for you.

```
$ :server connect
```

Connect to Neo4j

Connect URL
neo4j+s:// fa05846a.databases.neo4j.com

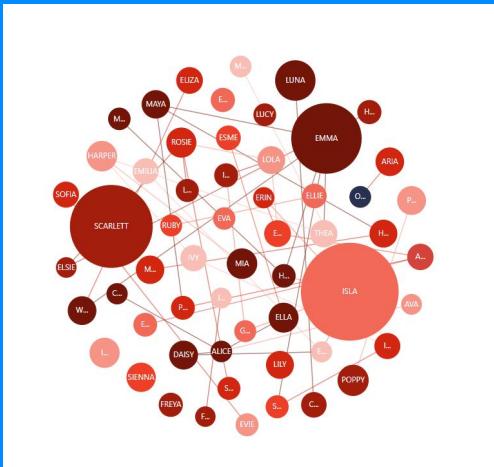
Authentication type
Username / Password

Username
neo4j

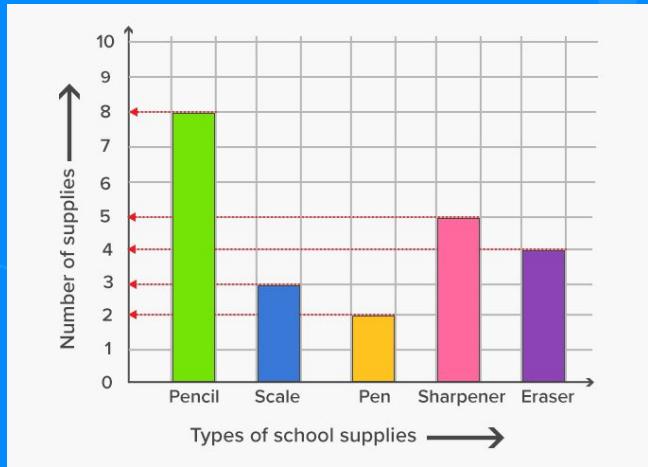
Password

Connect

What is a graph?

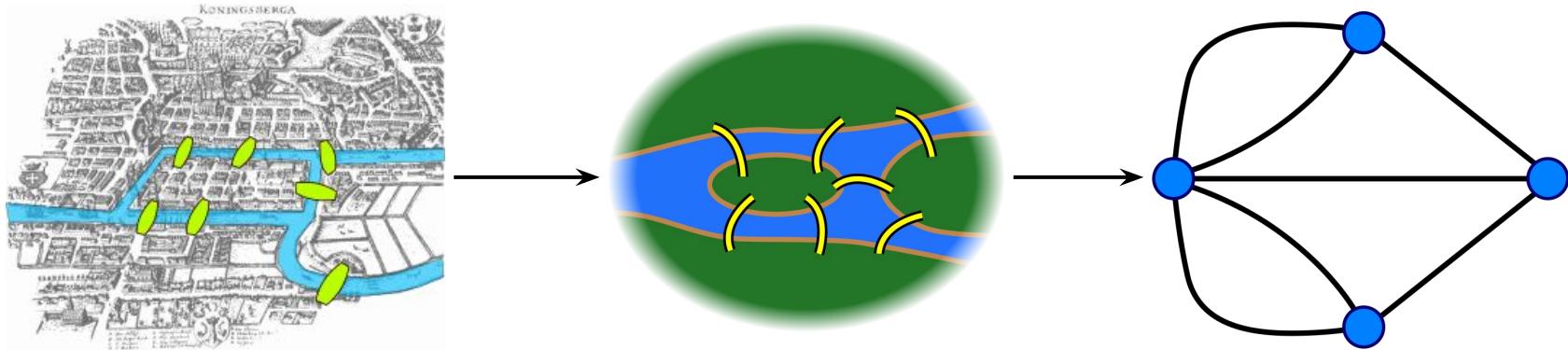


versus



A graph is...

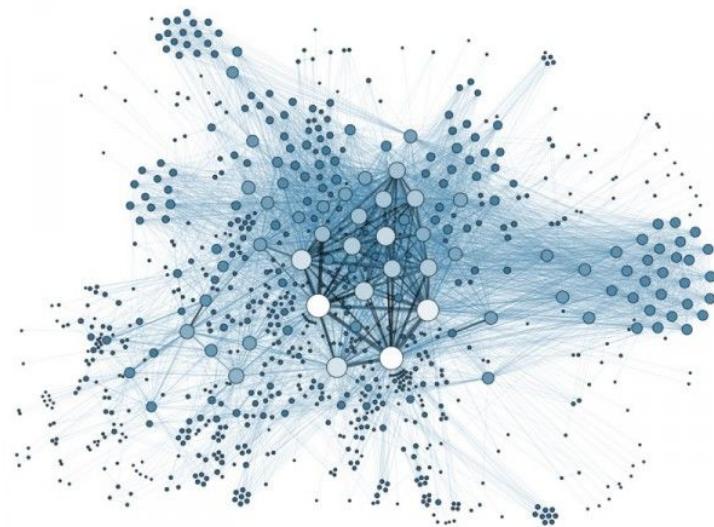
...a set of discrete entities, each of which has some set of relationships with the other entities



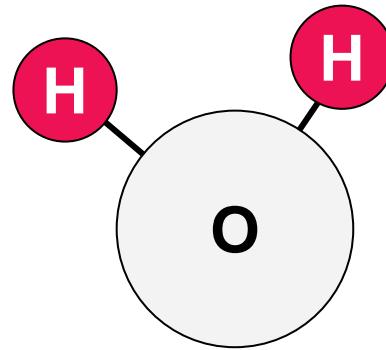
Seven Bridges of Königsberg problem. Leonhard Euler, 1735

Anything can be a graph - do you have examples too?

the Internet



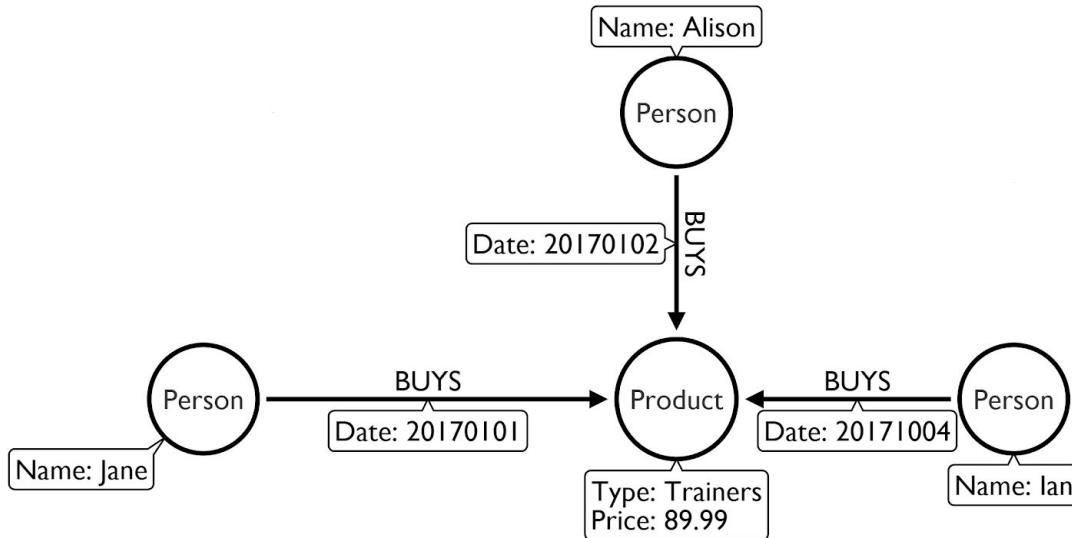
a water molecule



Why are graphs amazing?

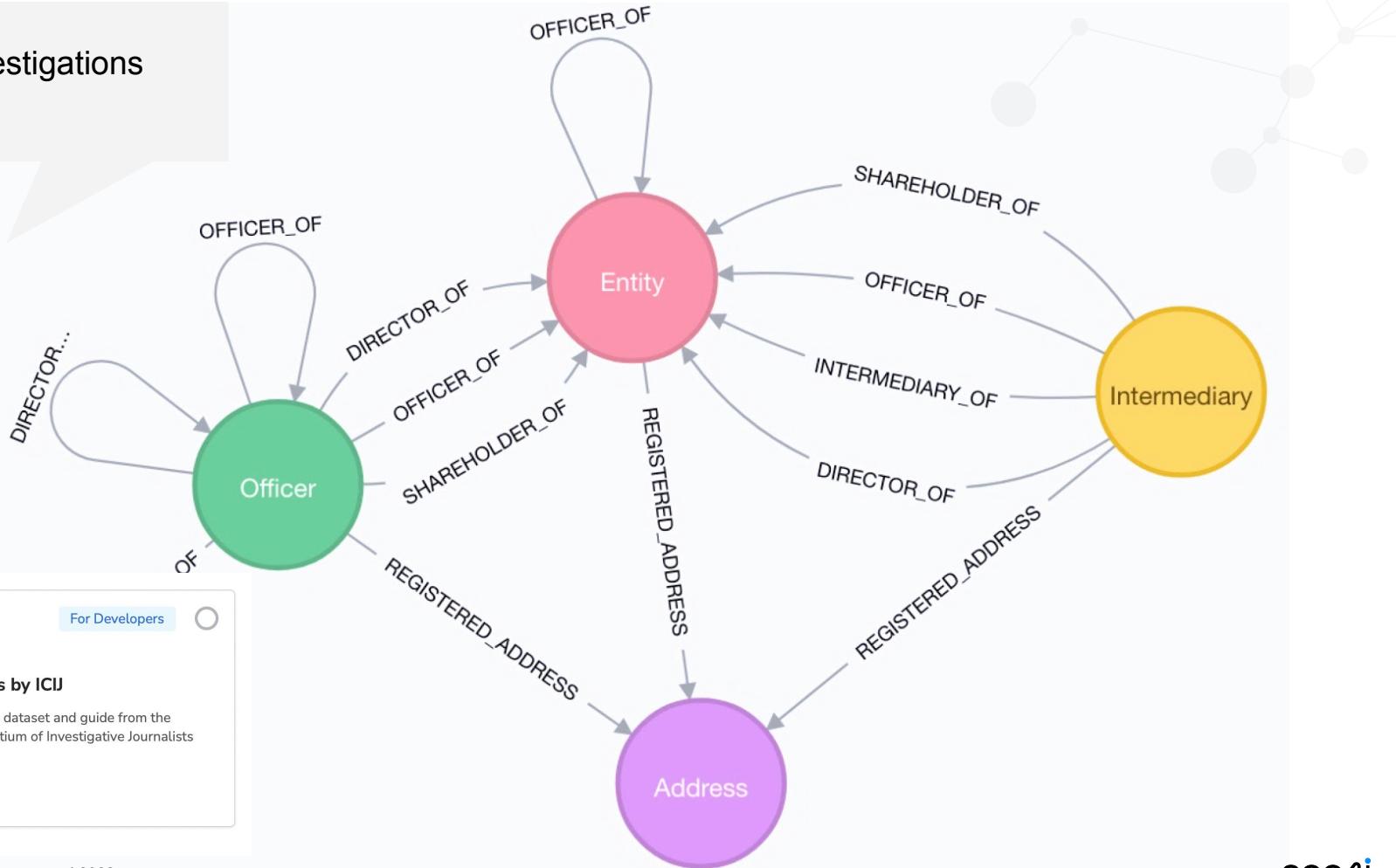


Follow the flow - buying trainers



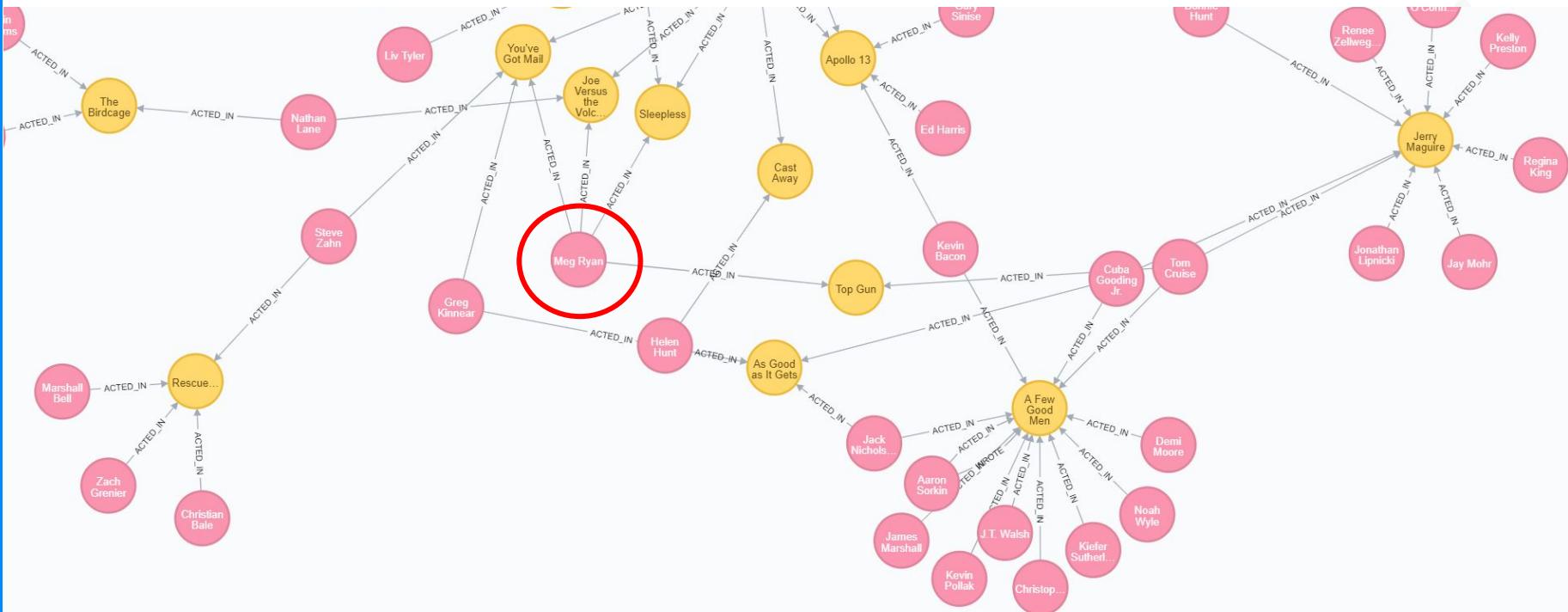
Panama, paradise, pandora papers: simple model, powerful outcome

The ICIJ Investigations data model...



*Roses are red,
facebook is blue,
No mutual friends,
So who are you?*

Friends of friends

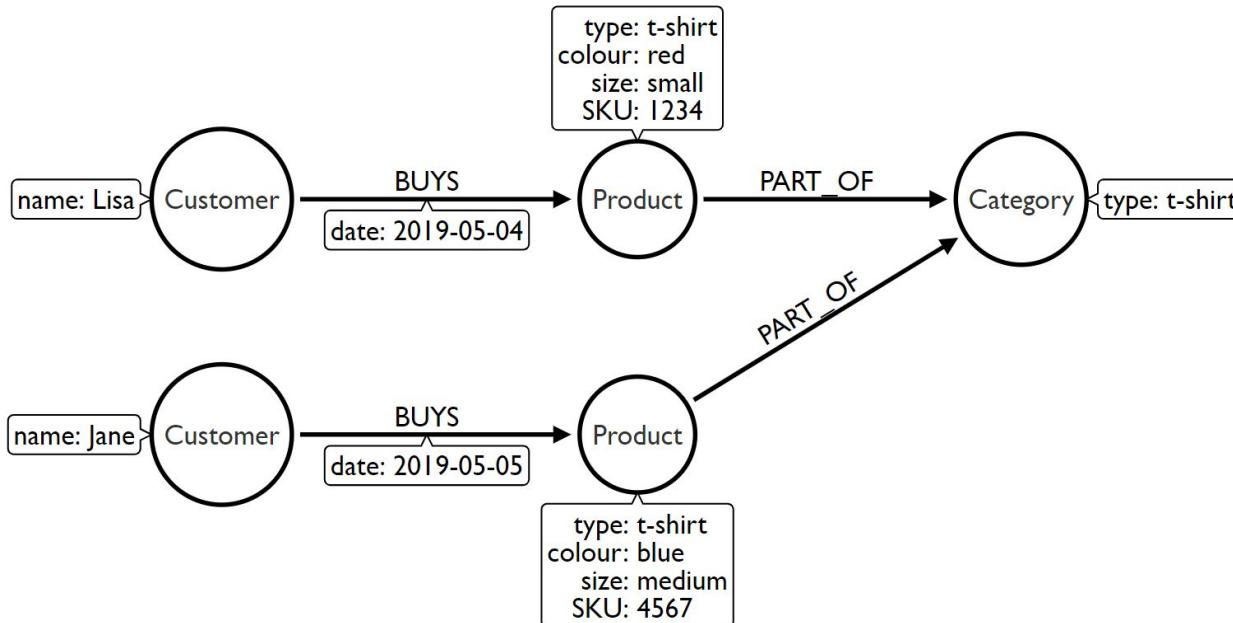


...or co-actors of co-actors

What are good graph scenarios?

Identifying good graph scenarios

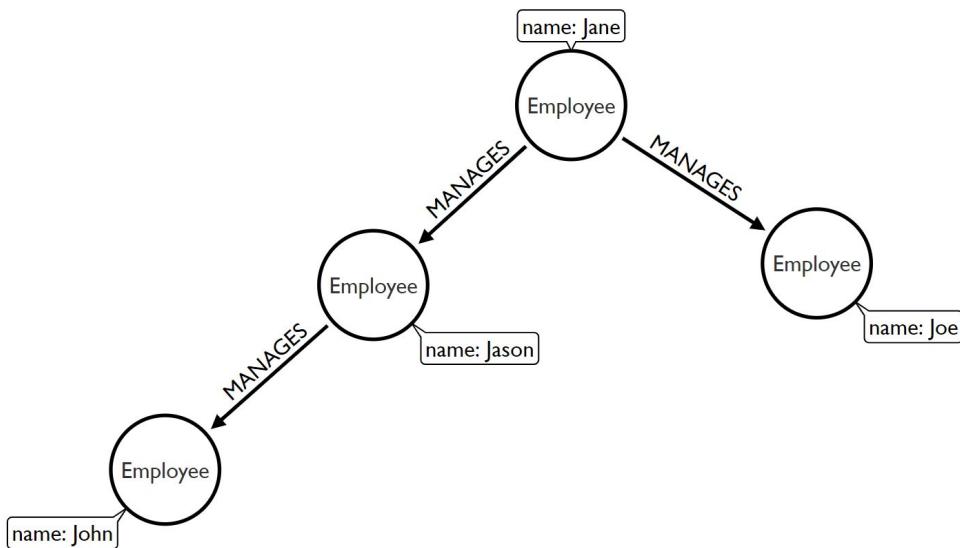
Scenario 1: Does our problem involve understanding relationships between entities?



- Recommendations
- Fraud detection
- Finding duplicates
- Data lineage
- Social Networks

Identifying good graph scenarios

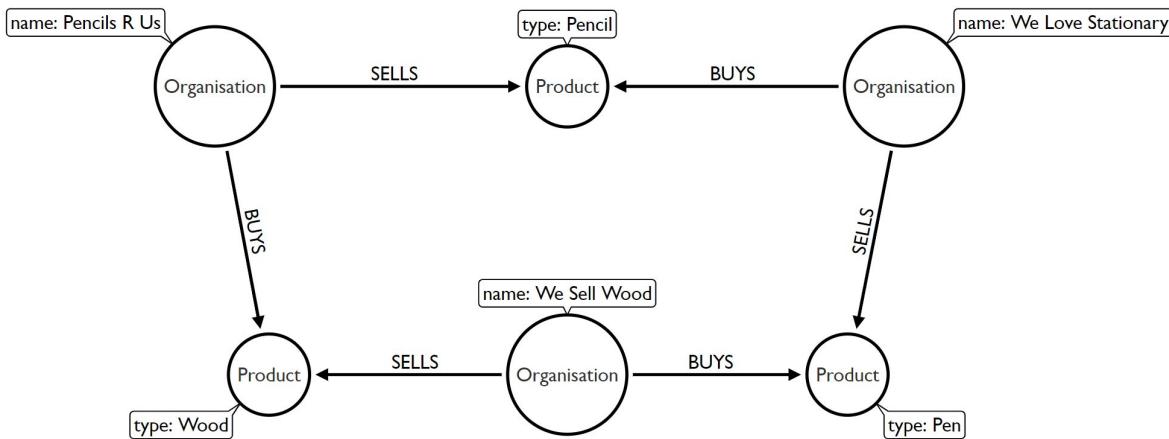
Scenario 2: Does the problem involve a lot of self-referencing to the same type of entity?



- Organisational hierarchies
- Access management
- Social influencers
- Friends of friends

Identifying good graph scenarios

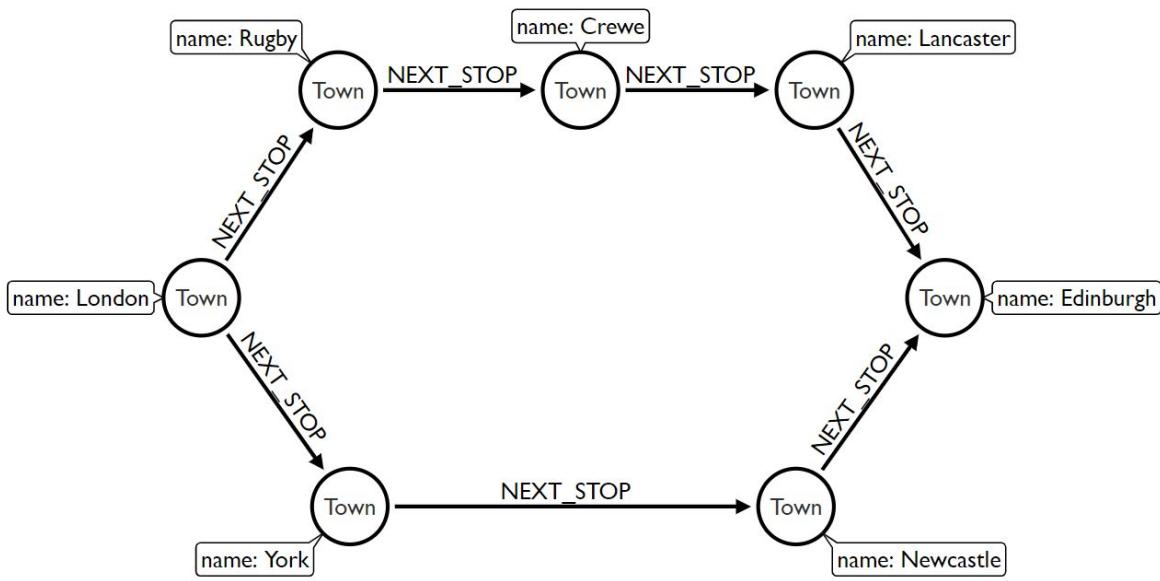
Scenario 3: Does the problem explore relationships of varying or unknown depth?



- Supply chain visibility
- Bill of Materials
- Network management
- Routing

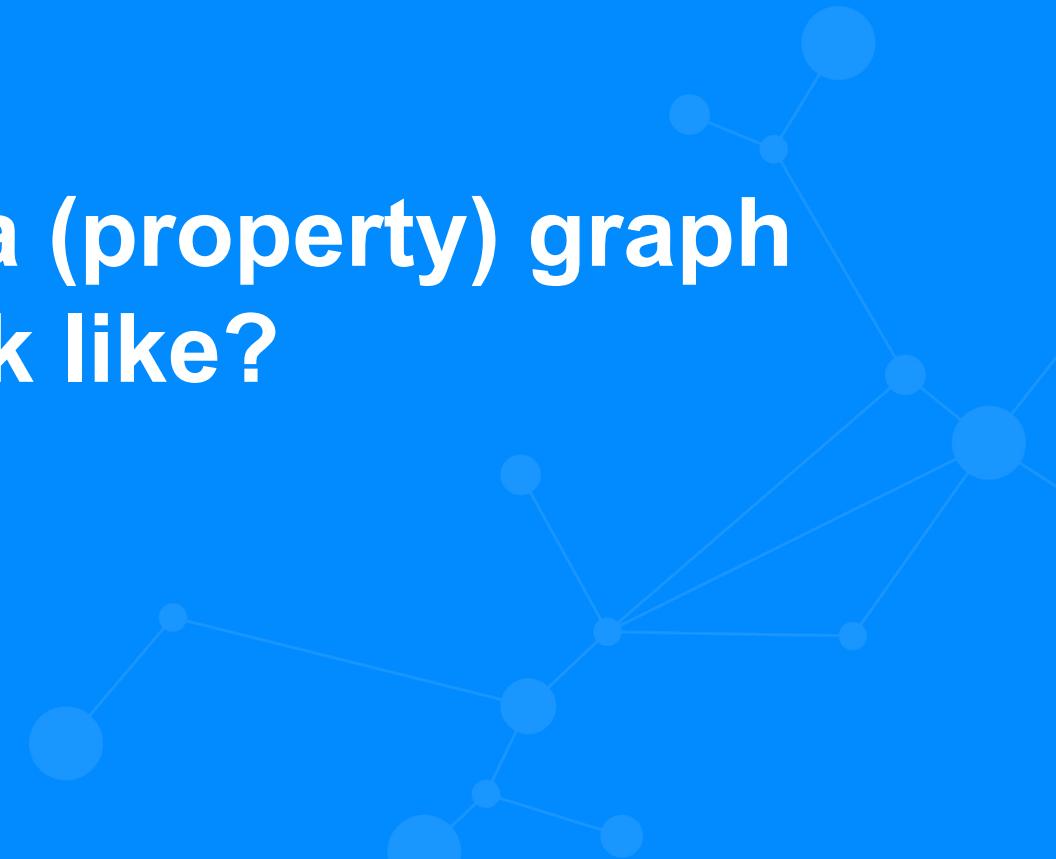
Identifying good graph scenarios

Scenario 4: Does our problem involve discovering lots of different routes or paths?



- Logistics and routing
- Infrastructure management
- Dependency tracing

So what does a (property) graph look like?



Graph components



Node (Vertex)

- The main data element from which graphs are constructed

Jane

A yellow circle containing the text "Jane".

bike

An oval containing the text "bike".

Graph components

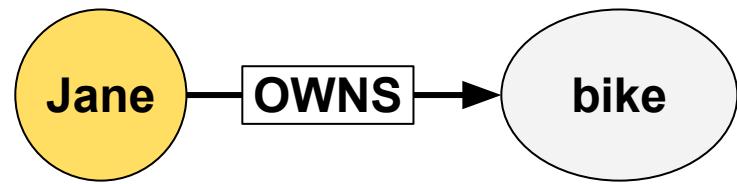


Node (Vertex)

- The main data element from which graphs are constructed

Relationship (Edge)

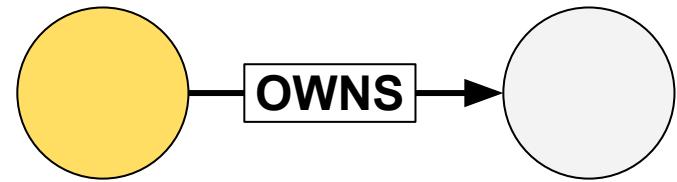
- A link between two nodes. Has:
 - Direction
 - Type
- A node without relationships is permitted. A relationship without nodes is not*



Property graph database

Node (Vertex)

Relationship (Edge)



Property graph database

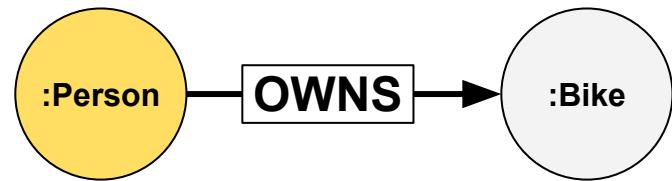


Node (Vertex)

Relationship (Edge)

Label

- Define node role (optional)



Property graph database

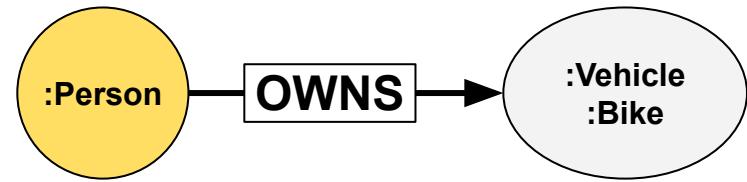


Node (Vertex)

Relationship (Edge)

Label

- Define node role (optional)
- Can have more than one



Property graph database

Node (Vertex)

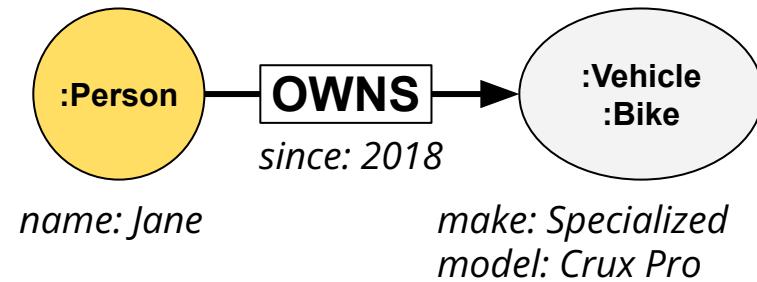
Relationship (Edge)

Label

- Define node role (optional)
- Can have more than one

Properties

- Enrich a node or relationship
- No need for nulls!



And now query the graph together!

Cypher

A **pattern**-matching query language made for graphs



Cypher



A **pattern** matching query language made for graphs

- Declarative
- Expressive
- Pattern-Matching

Cypher



A pattern matching query language made for graphs

- Declarative
- Expressive
- Pattern Matching

With ASCII ART

Legend

- Read**
- Write**
- General**
- Functions**
- Schema**
- Performance**
- Multidatabase**
- Security**

Syntax

Read query structure

```
[USE]
[WITH WHERE]
[OPTIONAL MATCH WHERE]
[WITH [ORDER BY] [SKIP] [LIMIT]]
RETURN [ORDER BY] [SKIP] [LIMIT]
```

MATCH ↗

```
MATCH (n:Person)-[:KNOWS]-(r:Person)
WHERE n.name = 'Alice'
```

Node patterns can contain labels and properties.

MATCH (n) ->(m)

Any pattern can be used in MATCH.

MATCH (n {name: 'Alice'}) ->(m)

Patterns with node properties.

MATCH p = (n) ->(m)

Assign a path to p.

OPTIONAL MATCH (n) -[r]->(n)

Optional pattern: nuls will be used for missing parts.

WHERE ↗

WHERE n.property => \$value

Use a predicate to filter. Note that WHERE is always part of a MATCH, OPTIONAL MATCH or WITH clause. Putting it after a different clause in a query will alter what it does.

WHERE EXISTS {
 MATCH (n) ->-(m) WHERE n.age = m.age
}

RETURN ↗

RETURN *

Return the value of all variables.

```
RETURN n AS columnName
Use alias for result column name.
```

RETURN DISTINCT n

Return unique rows.

ORDER BY n.property

Sort the result.

ORDER BY n.property DESC

Sort the result in descending order.

SKIP \$skipNumber

Skip a number of results.

LIMIT \$limitNumber

Limit the number of results.

SKIP \$skipNumber LIMIT \$limitNumber

Skip results at the top and limit the number of results.

RETURN count(*)

The number of matching rows. See Aggregating functions for more.

WITH ↗

```
MATCH (user)-[:FRIEND]-(friend)
WHERE user.name = $name
WITH user, count(friend) AS friends
WHERE friends > 10
RETURN user
```

The WITH syntax is similar to RETURN. It separates query parts explicitly, allowing you to declare which variables to carry over to the next part.

```
MATCH (user)-[:FRIEND]-(friend)
WITH user, count(friend) AS friends
ORDER BY friends DESC
SKIP 1
LIMIT 3
RETURN user
```

ORDER BY, SKIP, and LIMIT can also be used with WITH.

UNION ↗

MATCH (a)-[:KNOWS]-(b)

RETURN b.name

UNION

MATCH (a)-[:LOVES]-(b)

RETURN b.name

Returns the distinct union of all query results. Result column types and names have to match.

Operators ↗

General DISTINCT, .. []

Mathematical +, -, *, /, %, ^

Comparison =, <, <, >, >, IS NULL, IS NOT NULL

Boolean AND, OR, XOR, NOT

String +

List +, IN, [x], [x .. y]

Regular Expression =~

String matching STARTS WITH, ENDS WITH, CONTAINS

null ↗

- null is used to represent missing/undefined values.
- null is not equal to null. Not knowing two values does not imply that they are the same value. So the expression null = null yields null and not true. To check if an expression is null, use IS NULL.
- Arithmetic expressions, comparisons and function calls (except coalesce) will return null if any argument is null.
- An attempt to access a missing element in a list or a property that doesn't exist yields null.
- In OPTIONAL MATCH clauses, nulls will be used for missing parts of the pattern.

Patterns ↗

(n:Person)

Node with Person label.

(n:Person:Swedish)

Node with both Person and Swedish labels.

(n:Person {name: \$value})

Node with the declared properties.

()-[r {name: \$value}]-()

Matches relationships with the declared properties.

(n) -> -(m)

Relationship from n to m.

(n) -> (m)

Relationship in any direction between n and m.

(n:Person) ->-(n)

Maps ↗

{name: 'Alice', age: 38,

address: {city: 'London', residential: true}}

Literal maps are declared in curly braces much like property maps. Lists are supported.

WITH {person: {name: 'Anne', age: 25}} AS p

RETURN p.person.name

USE ↗

USE myDatabase

Select myDatabase to execute query, or query part, against.

```
USE neo4j
MATCH (n:Person)-[:KNOWS]-(m:Person)
WHERE n.name = 'Alice'
MATCH query executed against neo4j database.
```

SHOW FUNCTIONS and PROCEDURES ↗

SHOW FUNCTIONS

Listing all available functions.

SHOW PROCEDURES EXECUTABLE YIELD name

List all procedures that can be executed by the current user and return only the name of the procedures.

Labels

```
CREATE (n:Person {name: $value})
Create a node with label and property.
```

```
MERGE (n:Person {name: $value})
Matches or creates unique node(s) with the label and property.
```

SET n:\$property:Parent:Employee

Add label(s) to a node.

MATCH (n:Person)
Matches nodes labeled Person.

MATCH (n:Person)
WHERE n.name = \$value

Matches nodes labeled Person with the given name.

MATCH (n:Person)
Checks the existence of the label on the node.

labels(n)

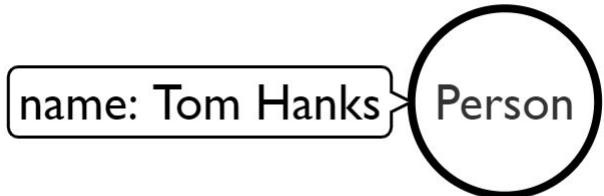
Labels of the node.

REMOVE n:Person

Remove the label from the node.

Use MATCH to retrieve nodes

```
//Match all nodes  
MATCH (n)  
RETURN n;
```



Use MATCH to retrieve nodes

```
//Match all nodes  
MATCH (n)  
RETURN n;  
  
//Match all nodes with a Person label  
MATCH (n:Person)  
RETURN n;
```



Use MATCH to retrieve nodes

```
//Match all nodes
```

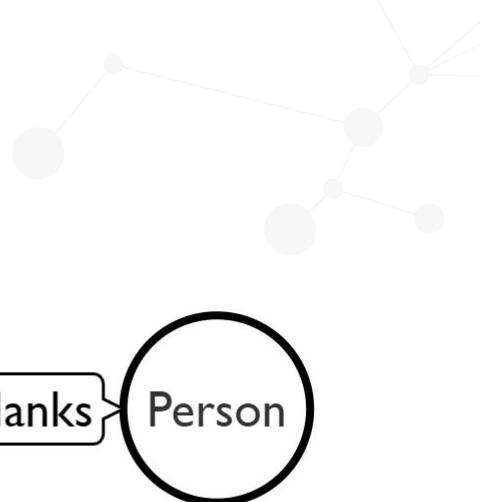
```
MATCH (n)  
RETURN n;
```

```
//Match all nodes with a Person label
```

```
MATCH (n:Person)  
RETURN n;
```

```
//Match all nodes with a Person label and property name is "Tom Hanks"
```

```
MATCH (n:Person {name: "Tom Hanks"})  
RETURN n;
```



Use MATCH and properties to retrieve nodes

```
//Return nodes with label Person and name property is "Tom Hanks" -  
Inline  
MATCH (p:Person {name: "Tom Hanks"}) //Only works with exact matches  
RETURN p;
```

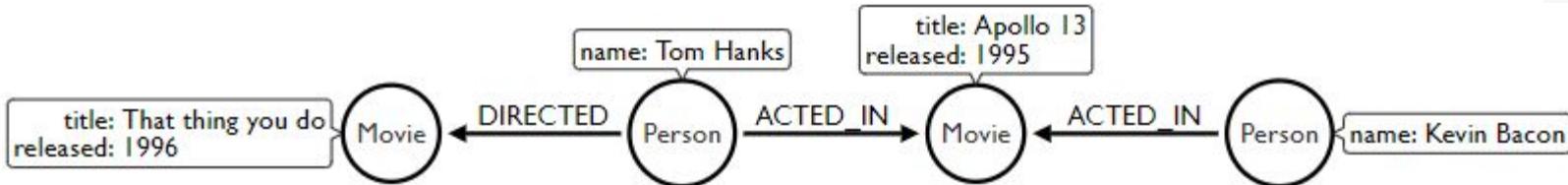
Use MATCH and properties to retrieve nodes

```
//Return nodes with label Person and name property is "Tom Hanks" -  
Inline  
MATCH (p:Person {name: "Tom Hanks"}) //Only works with exact matches  
RETURN p;  
  
//Return nodes with label Person and name property equals "Tom Hanks"  
MATCH (p:Person)  
WHERE p.name = "Tom Hanks"  
RETURN p;
```

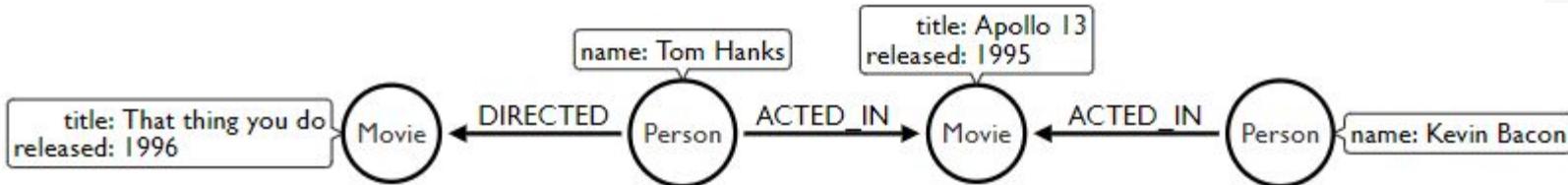
Use MATCH and properties to retrieve nodes

```
//Return nodes with label Person and name property is "Tom Hanks" -  
Inline  
MATCH (p:Person {name: "Tom Hanks"}) //Only works with exact matches  
RETURN p;  
  
//Return nodes with label Person and name property equals "Tom Hanks"  
MATCH (p:Person)  
WHERE p.name = "Tom Hanks"  
RETURN p;  
  
//Return nodes with label Movie, released property is between 1991 and  
1999  
MATCH (m:Movie)  
WHERE m.released > 1990 AND m.released < 2000  
RETURN m;
```

Extending the MATCH

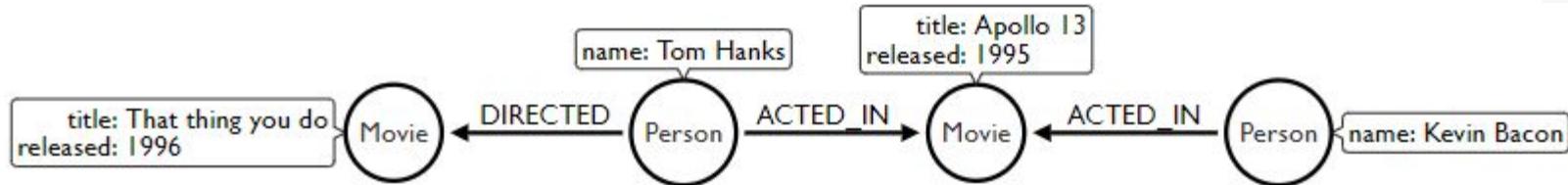


Extending the MATCH



```
MATCH (:Person {name:"Tom Hanks"}) -- (m:Movie)  
RETURN m.title;
```

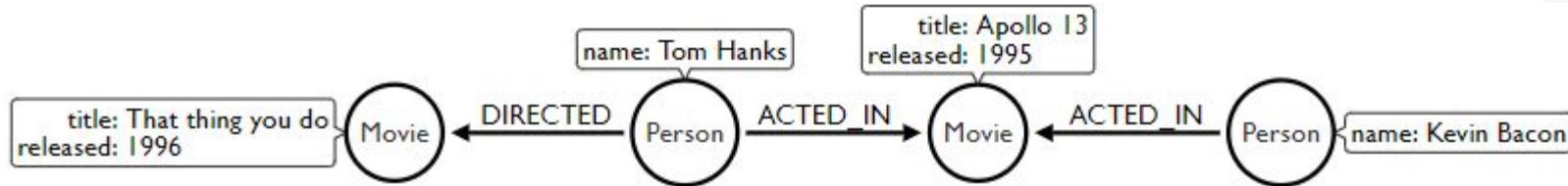
Extending the MATCH



```
MATCH (:Person {name:"Tom Hanks"})--(m:Movie)  
RETURN m.title;
```

```
//Find all the movies Tom Hanks directed and order by latest movie  
MATCH (:Person {name:"Tom Hanks"})-[:DIRECTED]->(m:Movie)  
RETURN m.title, m.released ORDER BY m.released DESC;
```

Extending the MATCH



```
MATCH (:Person {name:"Tom Hanks"})--(m:Movie)  
RETURN m.title;
```

```
//Find all the movies Tom Hanks directed and order by latest movie  
MATCH (:Person {name:"Tom Hanks"})-[:DIRECTED]->(m:Movie)  
RETURN m.title, m.released ORDER BY m.released DESC;
```

```
//Find all of the co-actors Tom Hanks have worked with  
MATCH (:Person {name:"Tom Hanks"})-->(:Movie)<- [:ACTED_IN] - (coActor:Person)  
RETURN coActor.name;
```

MERGE

```
//Uniquely create a person node called "Tom Hanks"  
MERGE (p:Person {name:"Tom Hanks"});
```



MERGE



```
//Create a person node called "Tom Hanks"
MERGE (p:Person {name:"Tom Hanks"}) ;

//Create an ACTED_IN relationship between "Tom Hanks" and "Apollo 13"
MATCH (p:Person {name:"Tom Hanks"}) , (m:Movie {title:"Apollo 13"})
MERGE (p) - [:ACTED_IN] -> (m) ;
```

Nodes and relationships at a glance



Description	Node	Relationship
Generic	()	-- --> - [] -
With a reference	(n)	- [r] -
With a node label or rel type	(:Person)	- [:ACTED_IN] -
With a label/type and an inline property	(:Person {name: 'Bob' })	- [:ACTED_IN {role: 'Dave' }] -
With a variable, label/type and an inline property	(p:Person {name: 'Bob' })	- [r:ACTED_IN {role: 'Rob' }] -

Stackoverflow Demo

:play sandbox/stackoverflow

What Else is There?

What else is there?

Connectors

- Spark
- Kafka
- JDBC

Drivers

- Python
- JavaScript
- Java
- .Net
- Go

Data Science

- Graph Data Science Library
- Bloom Visualization

Libraries / Integrations

- neo4j/graphql
- Spring Data Neo4j
- neosemantics (RDF)
- APOC (utility)

So how do I continue my graph journey?

A training class each week - Tuesdays, 3pm UTC

09 Mar: Getting Started with Neo4j Bloom

16 Mar: Build APIs with Neo4j GraphQL Library

23 Mar: Create a Knowledge Graph: A Simple ML Approach

Read all about it!
dev.neo4j.com/training

The image displays three cards representing training sessions from a series:

- Neo4j Training Series – Bloom**
WED Mar 09
04:00 pm GMT+1
We are running a series of training events! Second session: Getting started with Neo4j Bloom Want to get started with Neo4j Bloom? Keen to visualize your data? Then this is the session for you!
Training | Virtual | English
- Neo4j Training Series – GraphQL**
WED Mar 16
04:00 pm GMT+1
We are running a series of training events all week long! Third session: Full Stack GraphQL In The Cloud With Neo4j Aura, Next.js, & Vercel In this workshop we will build and deploy a full stack GraphQL application using Next.js.... Read more →
Training | Virtual | English
- Neo4j Training Series – Knowledge Graph**
WED Mar 23
05:00 pm GMT+1
We are running a series of training events! Fourth session: Graph Databases versus SQL for Data Science: Identifying 'Graph-y' Problems in Your Data A frequent question from data scientists is "why would I want to use a graph database when... Read more →
Training | Virtual | English

Continue your journey

Free online training and certification:

- dev.neo4j.com/learn
- dev.neo4j.com/datasets

How to, best practices, hands on and community stories:

- dev.neo4j.com/videos

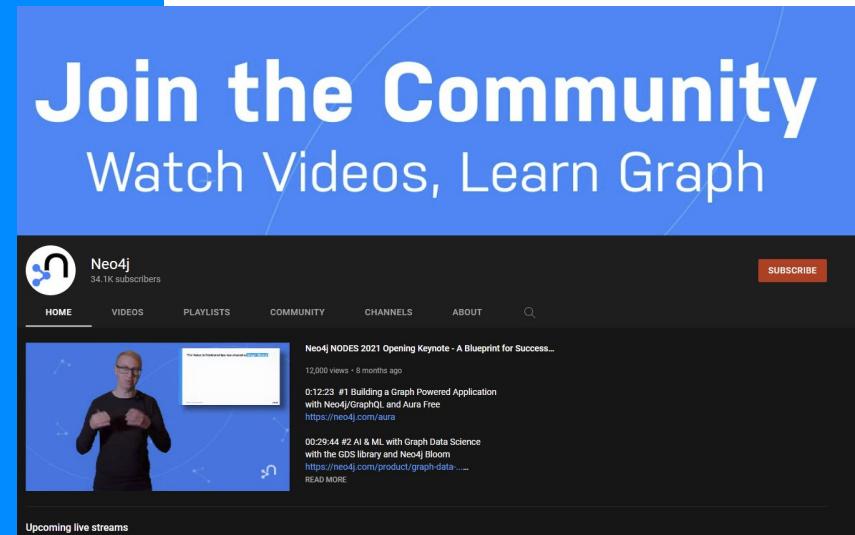
Come say hello :)

- dev.neo4j.com/chat
- dev.neo4j.com/forum

LEARN WITH GRAPHACADEMY

Free, Self-Paced, Hands-on Online Training

Learn how to build, optimize and launch your Neo4j project, all from the Neo4j experts.





Michael Hunger

Developer Relations



michael@neo4j.com

@mesirii

Join the conversation at dev.neo4j.com/forum