An Introduction to Neo4j

Agenda

1. What is a Graph Database?
2. What is Neo4j?
3. Graph DB concepts
4. Cypher – Demo/Hands on (!)

Some assumptions

- You have an instance of Neo4j Aura running
- You can connect to the instance
  - (*The test for this was in the document you received from us!*)

What is a Graph Database?

neo4j

What is a database?

Database: a structured set of data held in a computer, especially one that is accessible in various ways.

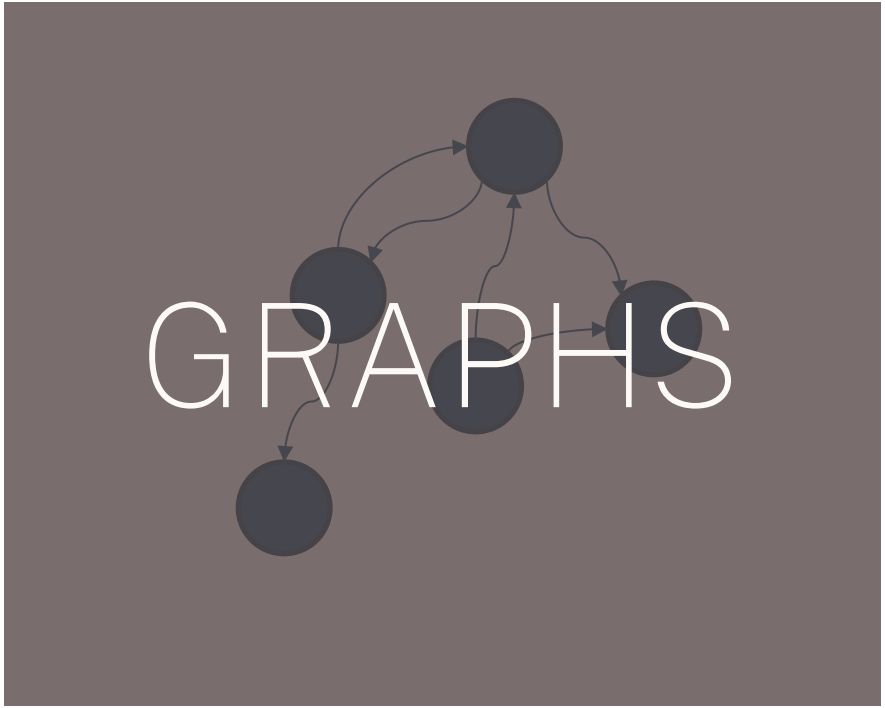
Relational

SQL Server
MySQL
Postgres
Oracle

NoSQL

Neo4j
Redis
MongoDB
Couchbase

What is a Graph?

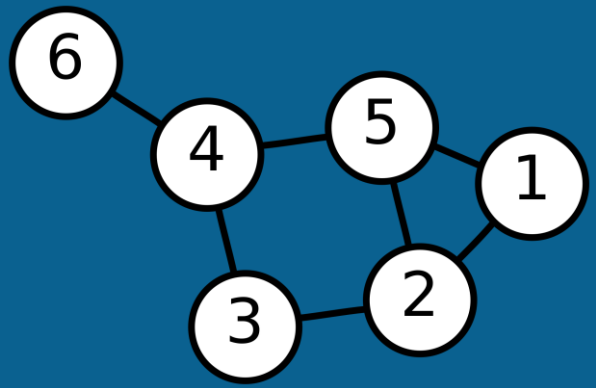


Graph Theory?

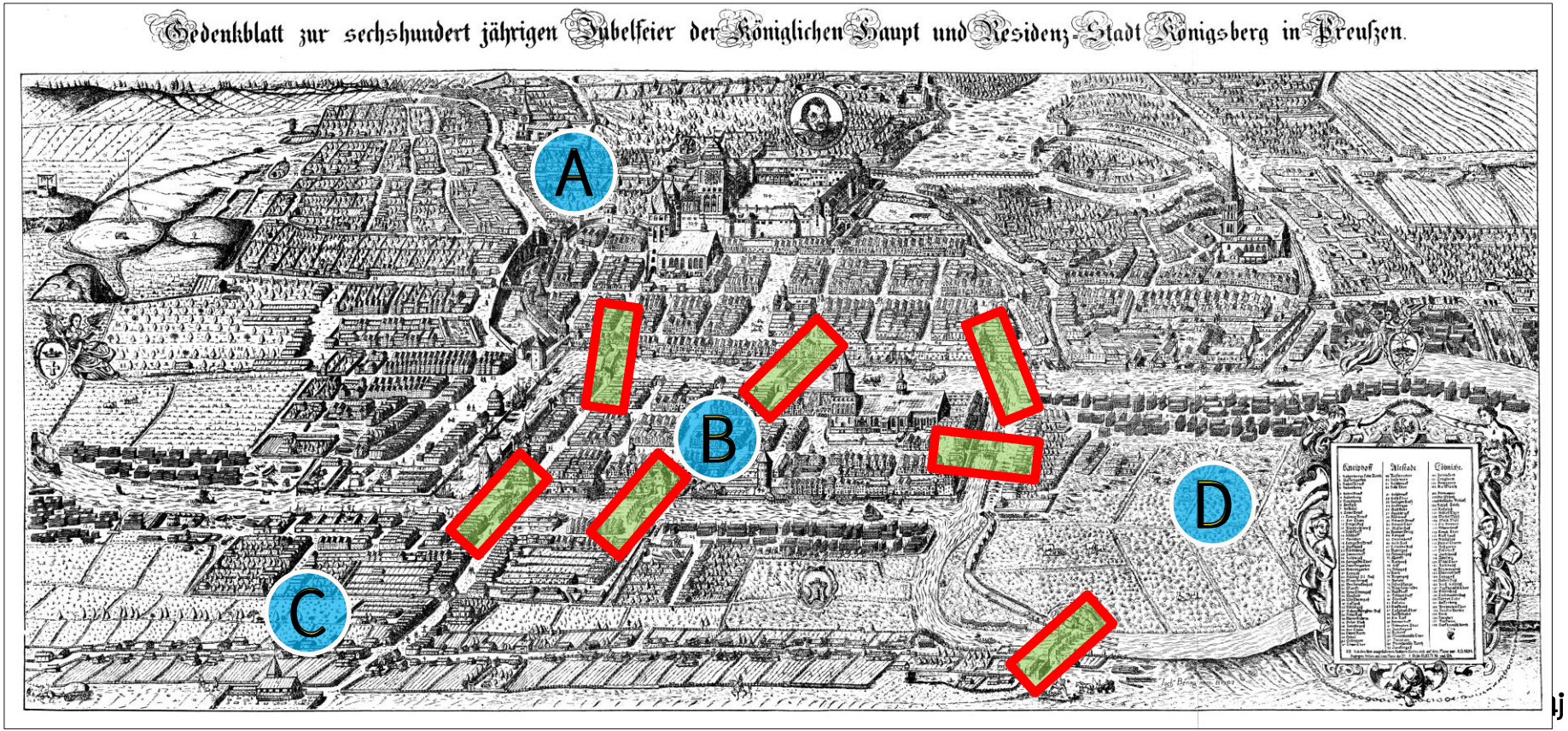
1736

In mathematics, and more specifically in graph theory, a graph is a structure amounting to a set of objects in which some pairs of the objects are in some sense "related".

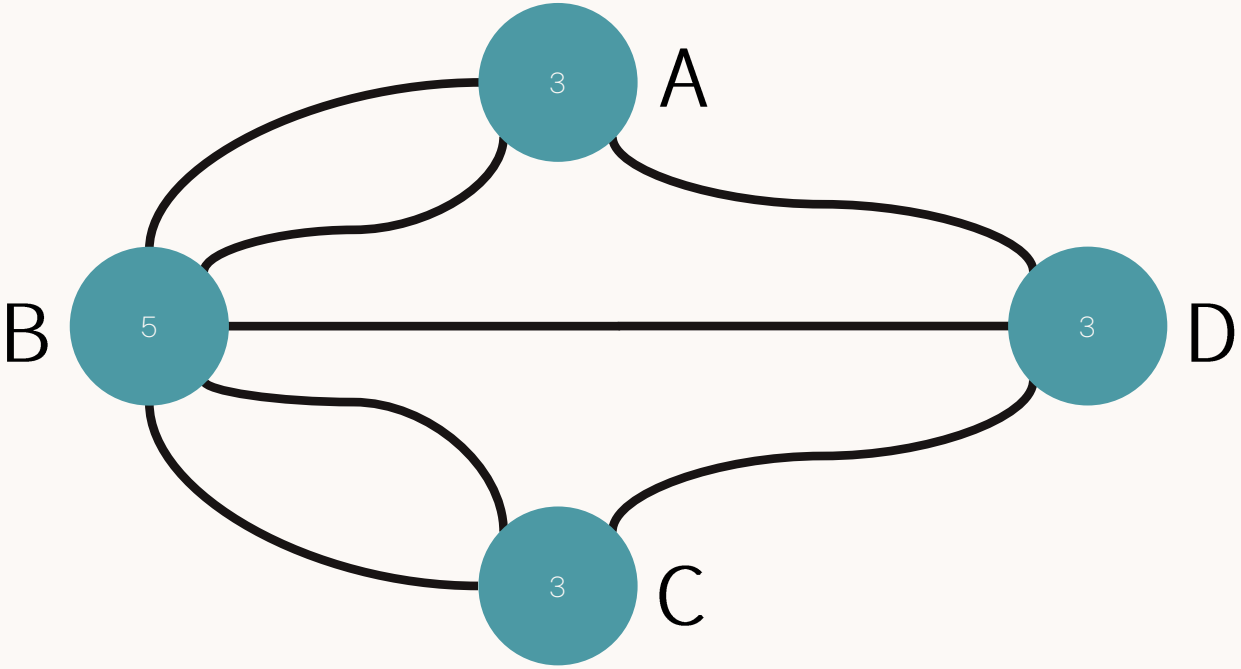
- Wikipedia



The Seven Bridges of Königsberg



The Seven Bridges of Königsberg



So… Graph Database?

Database: a structured set of data held in a computer, especially one that is accessible in various ways.

Graph Database: uses graph structures for semantic queries with nodes, relationships and properties to represent and store data.

A Graph Database is a database designed to treat relationships between the data as equally important to the data itself

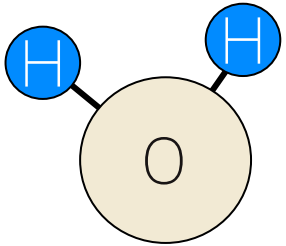
- Neo4j

neo4j

Anything can be a graph

The Public Internet

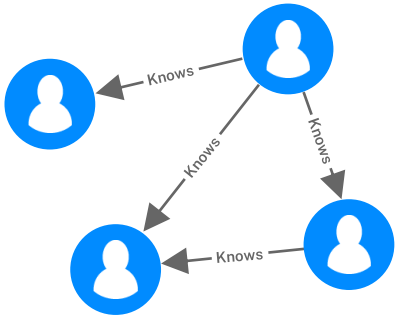
A water molecule



Social media

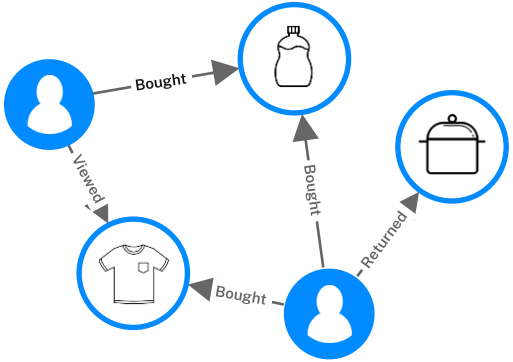
The metro

Connections in data are as valuable as the data itself



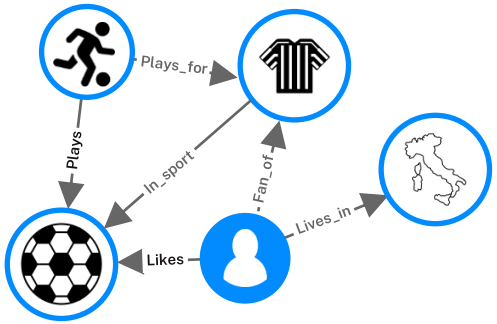
Networks of People

E.g., Employees, Customers, Suppliers, Partners, Influencers



Transaction Networks

E.g., Risk management, Supply chain, Payments



Knowledge Networks

E.g., Enterprise content, Domain specific content, eCommerce content

Neo4j

The Leading Graph Platform for Today's Intelligent Applications

The Forrester Wave™ for Graph Data Platforms, Q4 2020.

"Neo4j is the clear market leader in the graph space.

It has the most users, it uses and drives a widely adopted query language. In many respects, it has consistently been a lot more innovative than its competitors."

June, 2020
- Bloor InBrief

Bloor

#1 Most Popular Graph Database with Developers

Rank Dec 2023	Rank Nov 2023	Rank Dec 2022	DBMS	Database Model	Score Dec 2023	Score Nov 2023	Score Dec 2022
1.	1.	1.	Neo4j	Graph	49.99	+0.30	-7.34
2.	2.	2.	Microsoft Azure Cosmos DB	Multi-model	34.54	+0.43	-3.41
3.	3.	3.	Aerospike	Multi-model	7.18	-0.04	+0.54
4.	4.	4.	Virtuoso	Multi-model	5.27	-0.34	-0.68
5.	5.	5.	ArangoDB	Multi-model	4.43	-0.11	-0.92
6.	6.	6.	OrientDB	Multi-model	4.10	+0.32	-0.47
7.	7.	↑ 19.	Memgraph	Graph	3.18	+0.08	+2.13
8.	8.	↑ 9.	GraphDB	Multi-model	2.77	+0.03	+0.31
9.	↑ 10.	↓ 8.	Amazon Neptune	Multi-model	2.76	+0.28	-0.13
10.	↓ 9.	↑ 17.	NebulaGraph	Graph	2.67	+0.13	+1.52
11.	↑ 12.	↓ 7.	JanusGraph	Graph	2.30	+0.20	-0.69
12.	↓ 11.	↑ 13.	Stardog	Multi-model	2.28	-0.02	+0.62
13.	↑ 14.	↓ 12.	Dgraph	Graph	2.00	+0.09	+0.28
14.	↓ 13.	↓ 10.	TigerGraph	Graph	1.97	-0.02	-0.11

TOP 20

200k+
Developers

DB-ENGINES

The Neo4j database ecosystem

neo4j

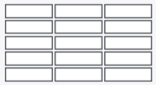
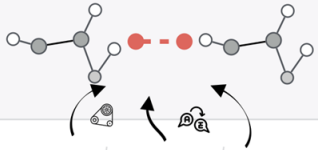
Neo4j is a database - ACID

- Atomicity, Consistency, Isolation, Durability
- A transaction succeeds as a whole or fails as a whole.
- This differentiates Neo4j from other graph database- and NOSQL vendors.
- Neo4j offers the same guarantees as an Oracle, SQL*Server, ...

Neo4j is a database - Secret Ingredient(s)

- Graph Native

Native vs Non-Native

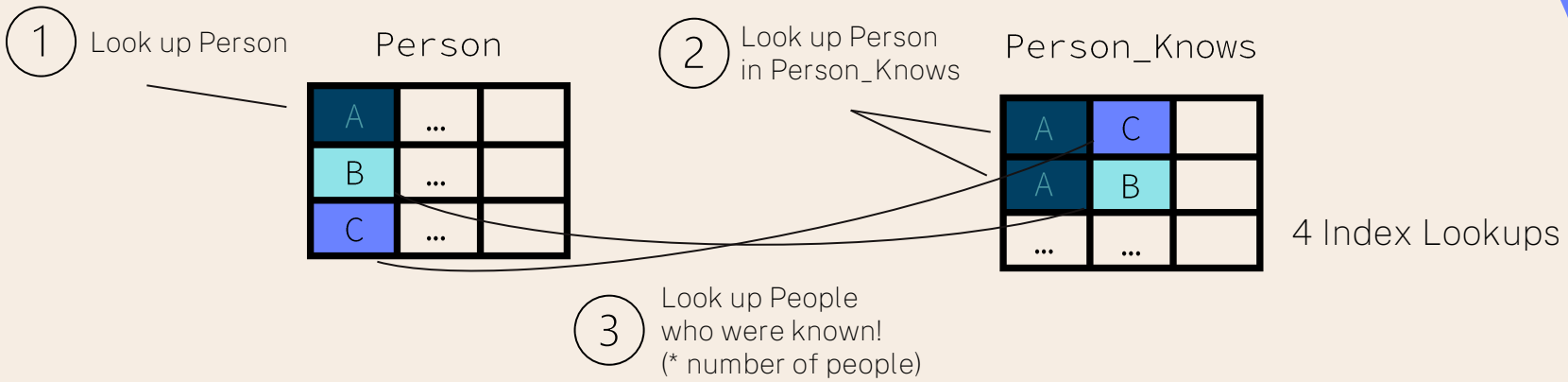
	Native Graph DB	Non-Native Graph DB	RDBMS
Visualization			
Queries	Cypher (graphs)-[are]->(everywhere)	Cypher/Gremlin /Proprietary	SQL
Processing			
Storage			

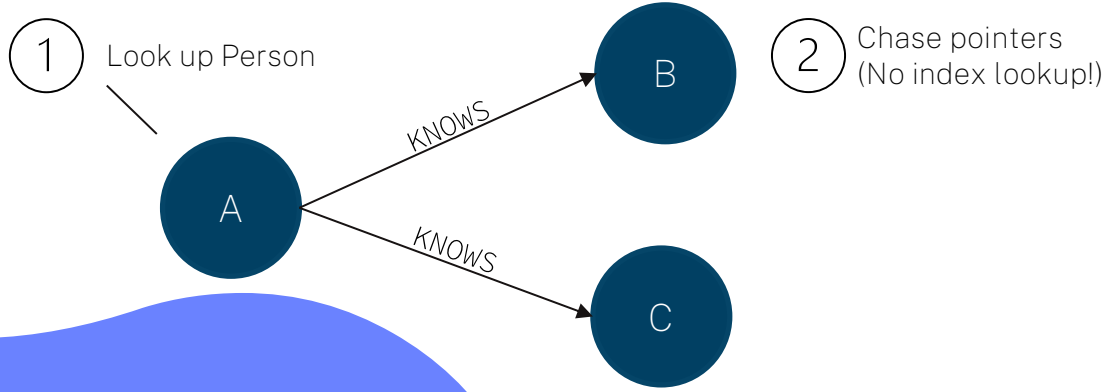
Optimized for graph workloads

Neo4j is a database - Secret Ingredient(s)

- Graph Native
- Index Free Adjacency

Index-Free Adjacency?



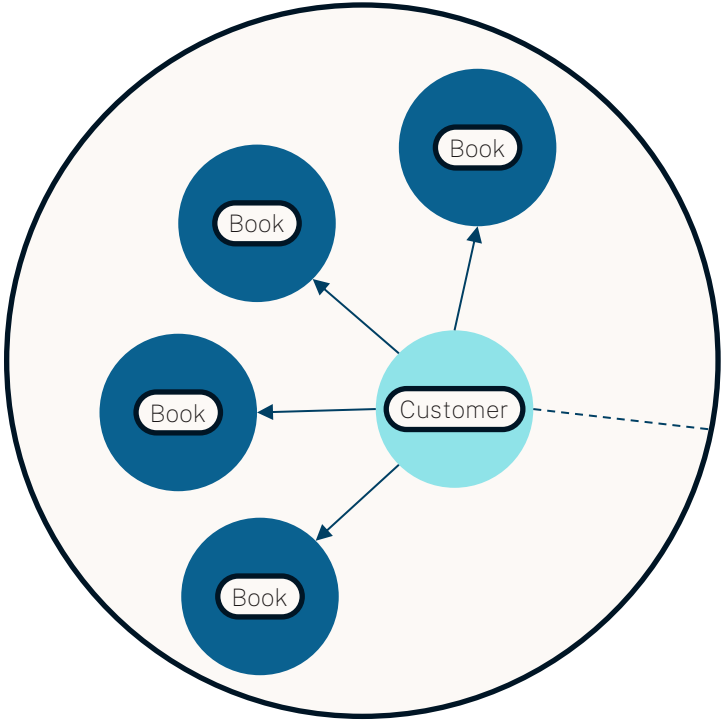


1 Index Lookup
(no matter HOW many friends)

Neo4j is a database - Secret Ingredient(s)

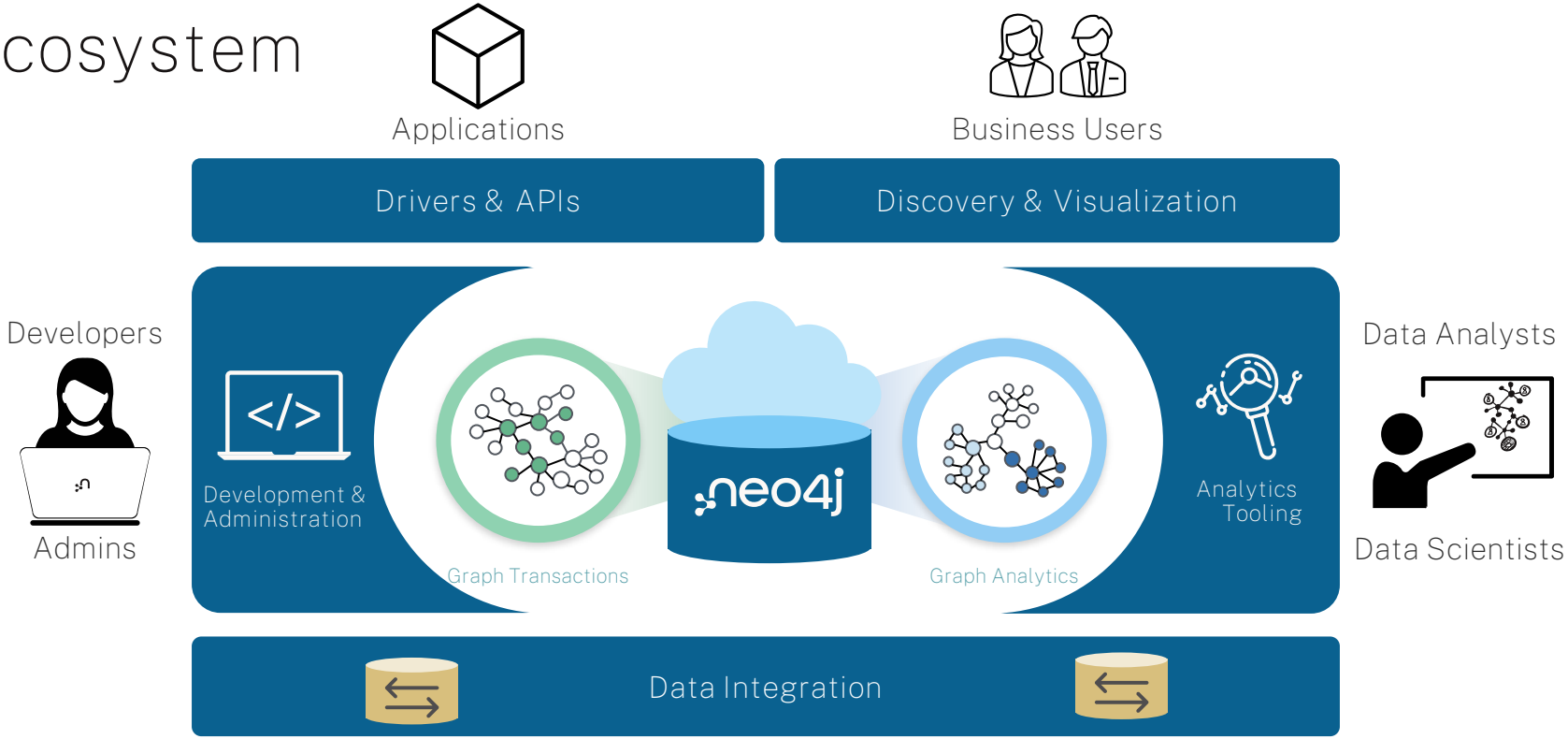
- Graph Native
- Index Free Adjacency
- Graph Locality

Neo4j is a database - Graph Locality



- Graph can be 100s of millions of nodes
- You query small areas
  - Local areas

Ecosystem


Applications
Business Users
Drivers & APIs
Discovery & Visualization
Developers
Admins
Development & Administration
Graph Transactions
neo4j
Graph Analytics
Analytics Tooling
Data Analysts
Data Scientists
Data Integration

Property Graph Components

Good news!

you only need to know

4

things

Graph components

Node (Vertex)

- The main data element from which graphs are constructed

Graph components

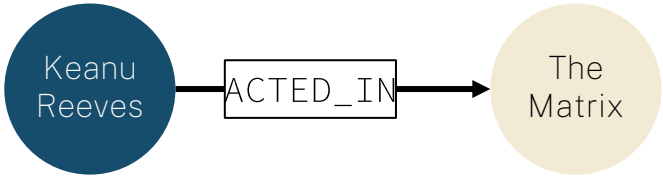
A node without relationships is permitted, a relationship without nodes is not

Node (Vertex)

- The main data element from which graphs are constructed

Relationship (Edge)

- A link between two nodes
  - Direction
  - Type



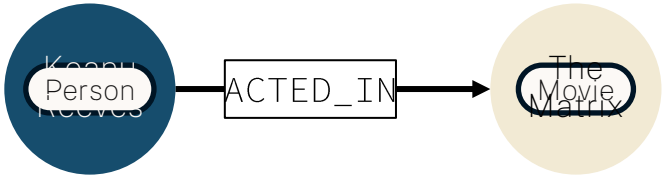
Property graph database

Node (Vertex)

Relationship (Edge)

Label

- Define node role (optional)



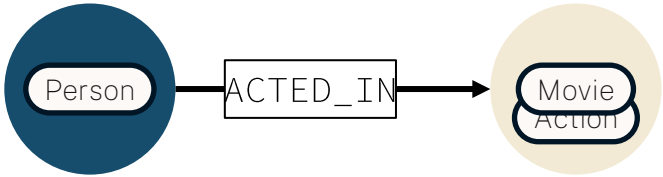
Property graph database

Node (Vertex)

Relationship (Edge)

Label

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



Property graph database

Node (Vertex)

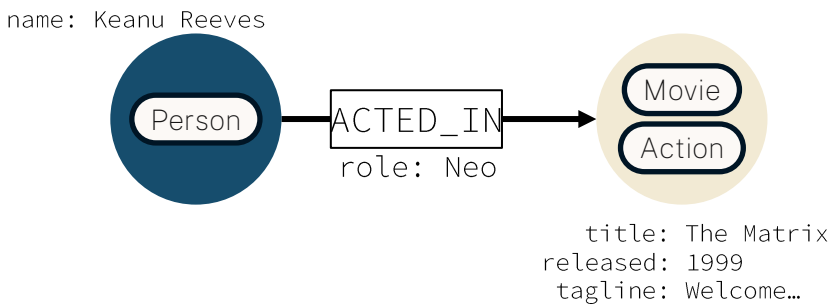
Relationship (Edge)

Label

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

Properties

- Enrich
  - nodes
  - relationships
- No need for nulls



Cypher

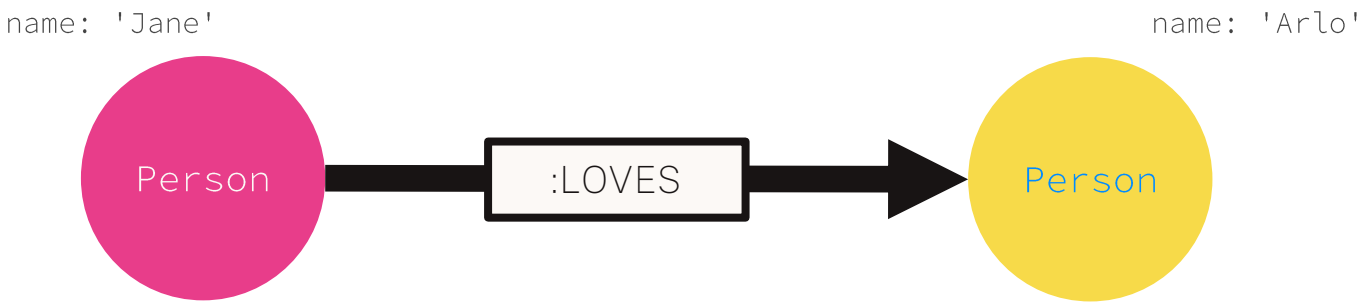
A new query language

neo4j

What is Cypher?

- A declarative query language for property graphs
- Uses a (limited) form of ASCII Art to allow you to visually describe the patterns in the graph

Cypher: powerful and expressive query language

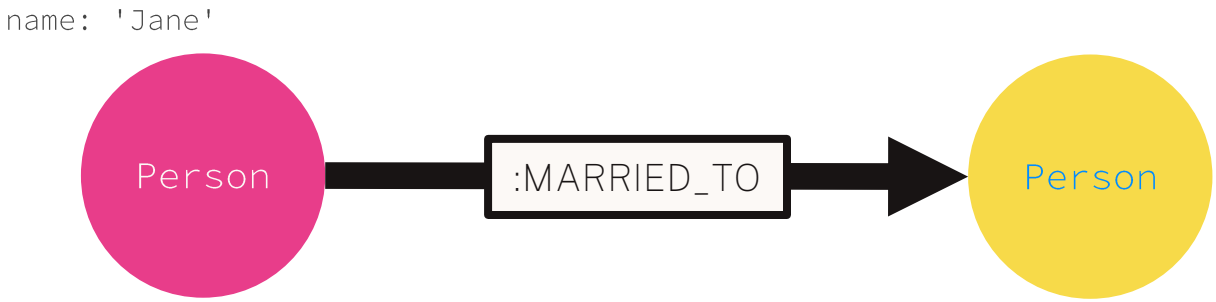


NODE RELATIONSHIP NODE

```
CREATE (:Person {name: 'Jane'}) -[:LOVES]-> (:Person { name: 'Arlo' })
```

Label Property Label Property

Cypher: Matching



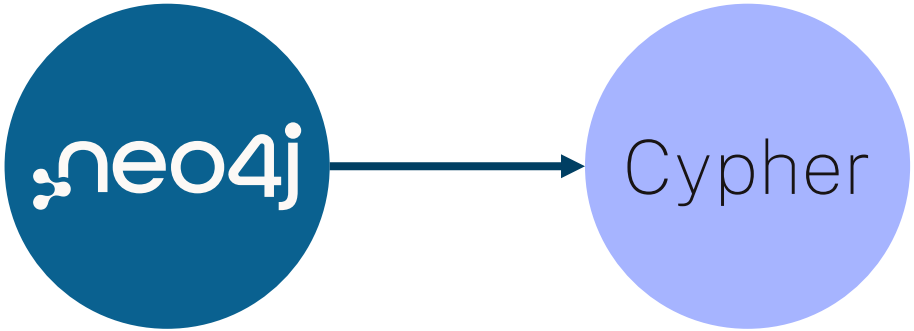
NODE RELATIONSHIP NODE

```
MATCH (p:Person {name: 'Jane'}) -[:MARRIED_TO]-> (spouse:Person)
```

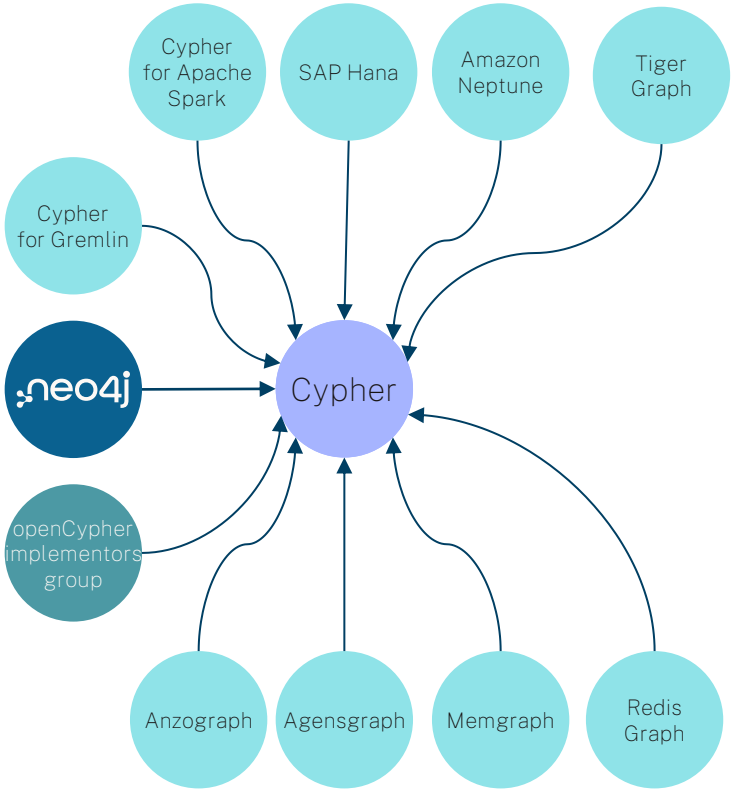
Variable Label Property Variable

```
RETURN p, spouse
```

Where is Cypher going?

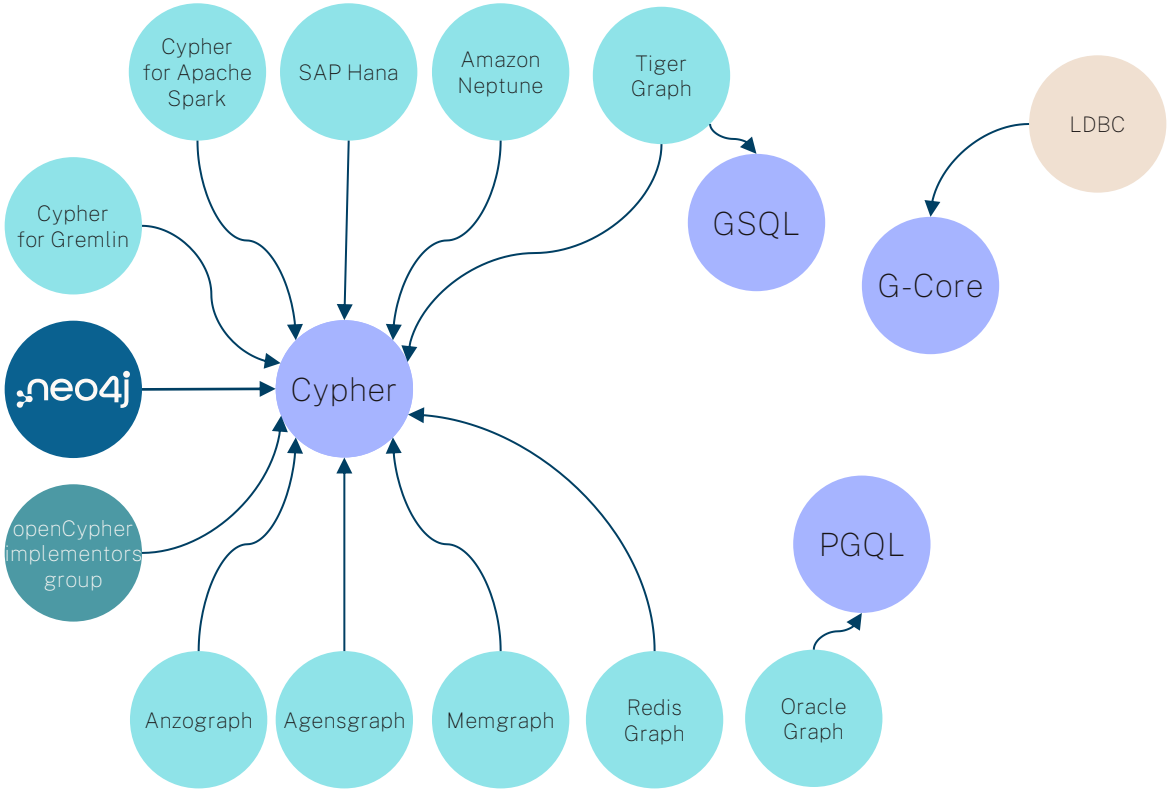


Where is Cypher going?



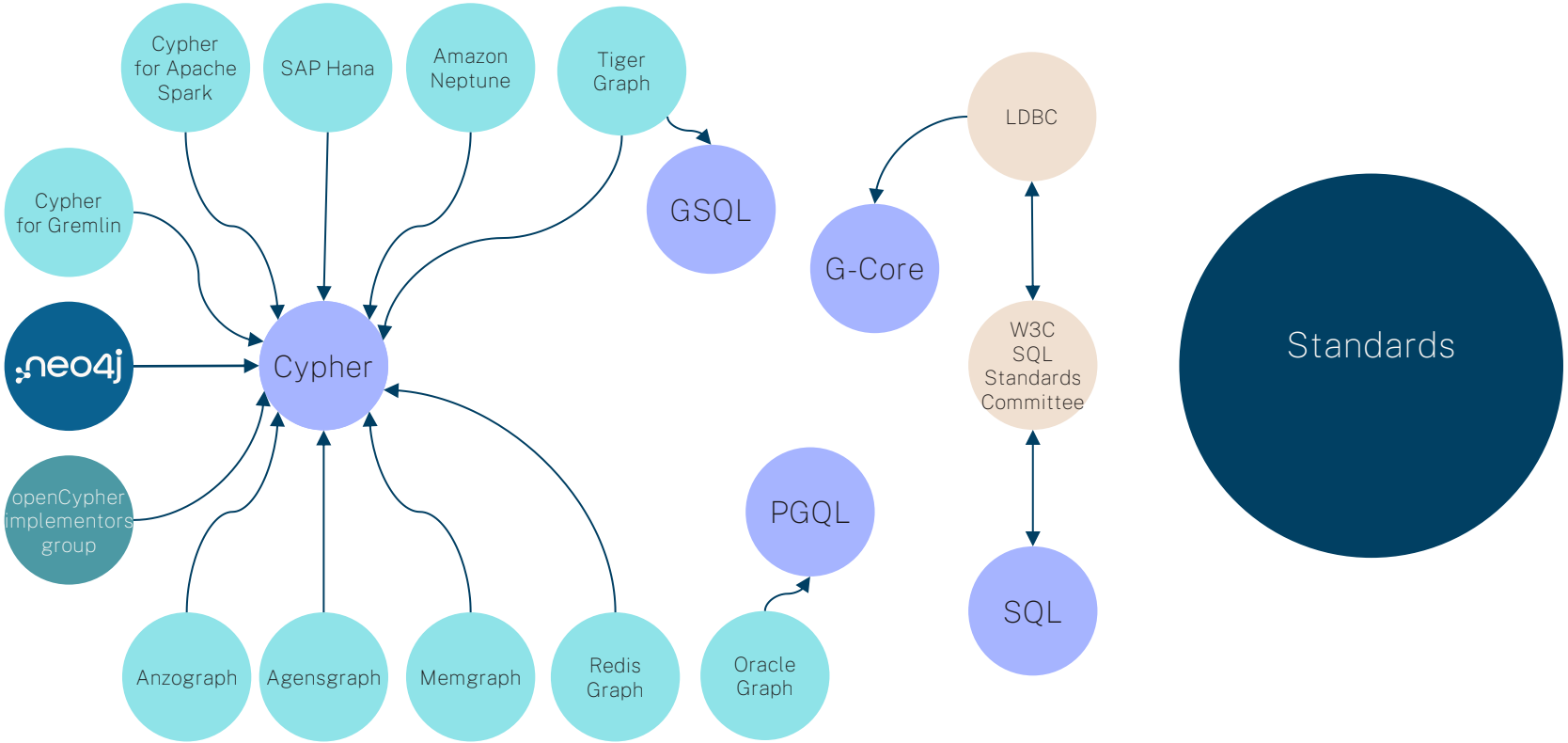
Cypher is not just Neo4j

Where is Cypher going?

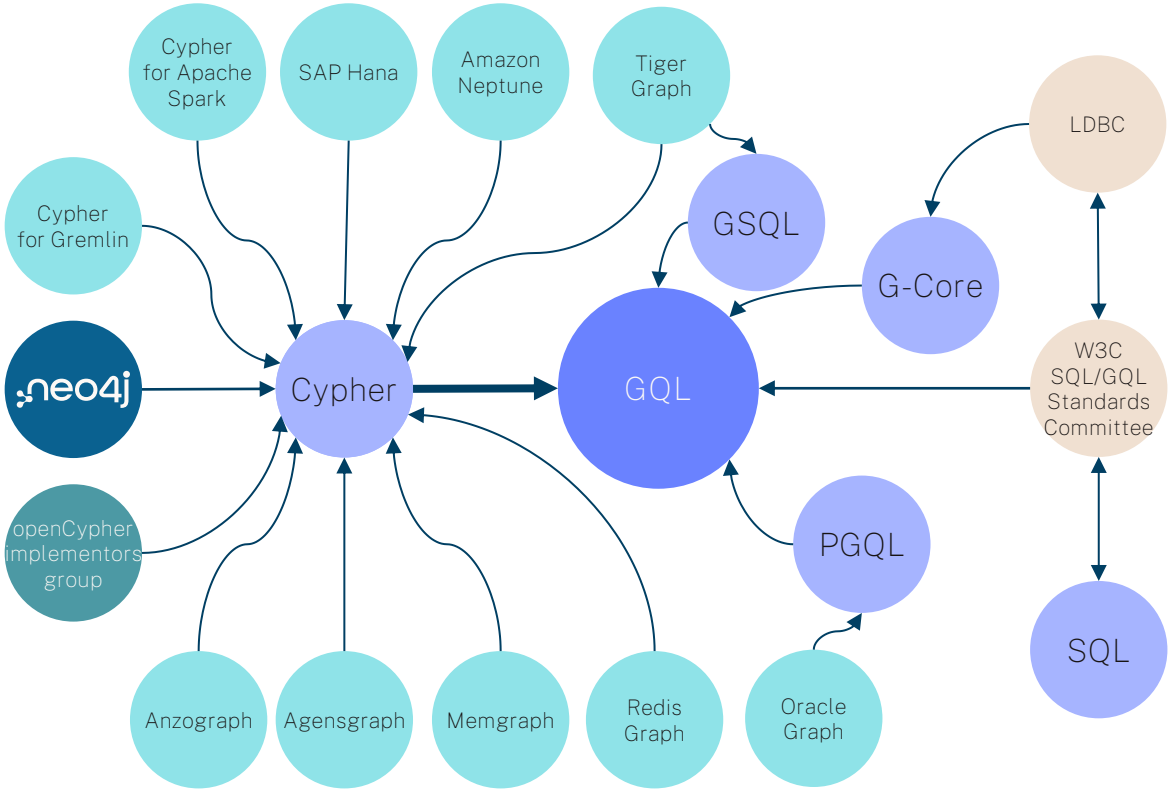


Different Solutions

Where is Cypher going?



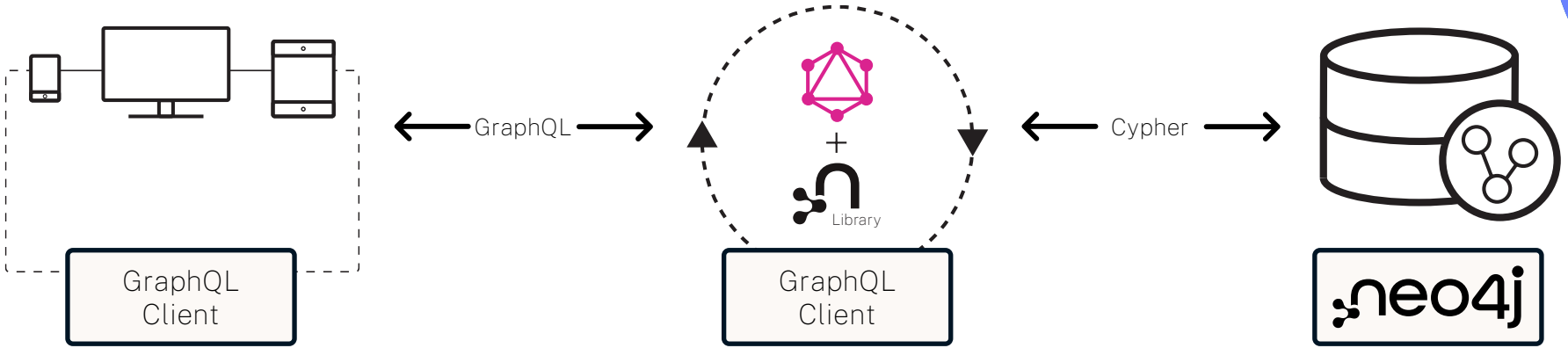
Where is Cypher going?



Standards Redefined

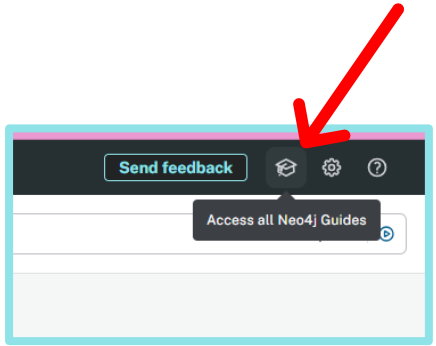
What about GraphQL?

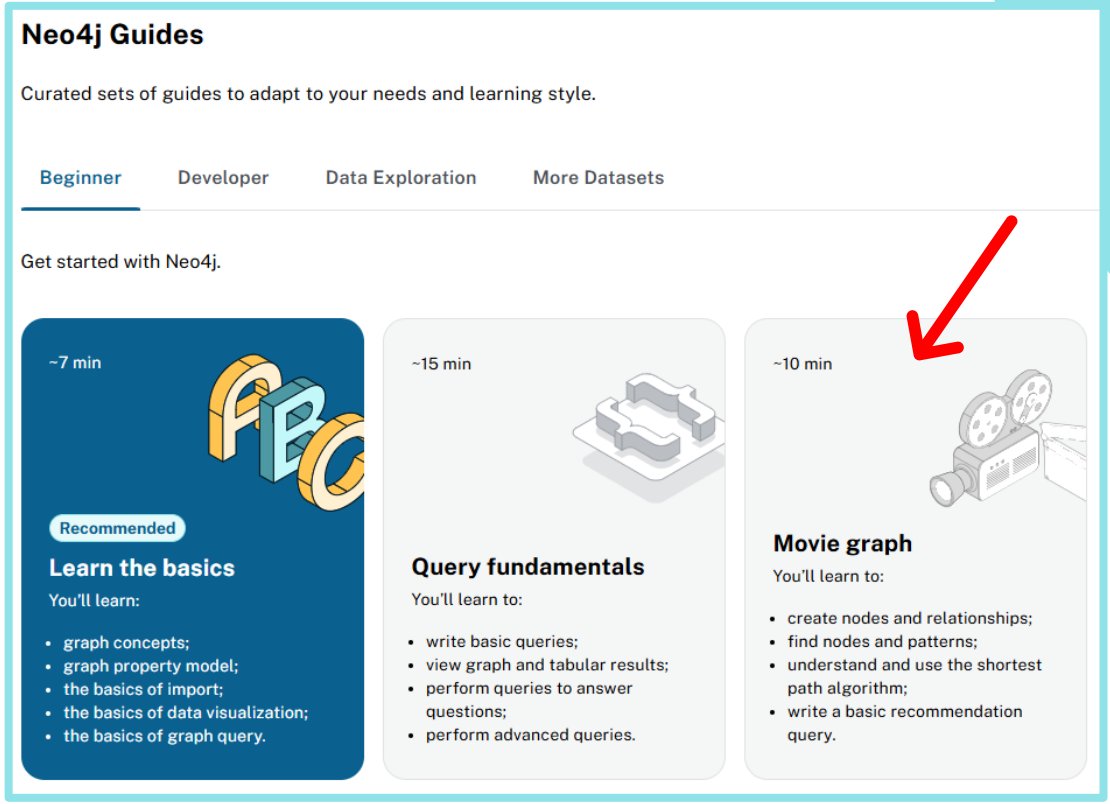
GQL ≠ GraphQL



Let’s load some data

Load the guides





Create the movie database

Movie graph

Step 1/8

Movies graph

This guide uses pop-cultural connections to teach you how to use Cypher to create and query data. You will explore connections between actors, directors, and movies, and also solve the famous Bacon-path.

Note that this guide assumes that you use an empty database as you will modify it along the way. You will learn how to delete the data you create towards the end of the guide as well.

What you will learn

This guide takes around 10 minutes and by the end of it you will be able to:

- Create nodes and relationships that connect them.
- Find individual nodes by using conditional matches.
- Find patterns in the graph.
- Understand and use the shortest path algorithm.
- Write a basic recommendation query.
- Delete all data.

Let's get started!

In the next step you are going to create the data you'll be working with.

Next

Step 2/8

Create the data

The Cypher `CREATE` clause is used to create data in your graph. If you are not familiar with Cypher syntax, you can try the *Query fundamentals* guide or see the Cypher Manual ↗.

You are going to create a dataset with `Person` and `Movie` nodes and different types of relationships to connect them.


```
1  CREATE CONSTRAINT movie_t
2  CREATE CONSTRAINT person_
3
4  MERGE (TheMatrix:Movie {title:'
5
6  MERGE (Keanu:Person {name:'Keanu
7  MERGE (Carrie:Person {name:'Car
8  MERGE (Laurence:Person {name:'La
9  MERGE (Hugo:Person {name:'Hugo
10 MERGE (LillyW:Person {name:'Lill
11 MERGE (LanaW:Person {name:'Lana
12 MERGE (JoelS:Person {name:'Joel
13
14 MERGE (Keanu)-[:ACTED_IN {rol
15 MERGE (Carrie)-[:ACTED_IN {rc
```

Wait for it to load

neo4j $ CREATE CONSTRAINT movie_title IF NOT EXISTS FOR (m:Movie) REQUIRE m.title IS

```
CREATE CONSTRAINT movie_title IF NOT EXISTS FOR (m:Movie) REQUIRE m.title IS UNIQUE
CREATE CONSTRAINT person_name IF NOT EXISTS FOR (p:Person) REQUIRE p.name IS UNIQUE
MERGE (TheMatrix:Movie {title:'The Matrix'}) ON CREATE SET TheMatrix.released=1999, TheMa
MERGE (TheMatrixReloaded:Movie {title:'The Matrix Reloaded'}) ON CREATE SET TheMatrixRelo
MERGE (TheMatrixRevolutions:Movie {title:'The Matrix Revolutions'}) ON CREATE SET TheMatr
MERGE (TheDevilsAdvocate:Movie {title:"The Devil's Advocate", released:1997, tagline:'Evi
MERGE (AFewGoodMen:Movie {title:'A Few Good Men'}) ON CREATE SET AFewGoodMen.released=199
MERGE (TopGun:Movie {title:'Top Gun'}) ON CREATE SET TopGun.released=1986, TopGun.tagline
```

Database information

Nodes (171)

* Movie Person

Relationships (253)

* ACTED_IN DIRECTED FOLLOWS PRODUCED REVIEWED WROTE

Property keys

born name rating released roles summary tagline title

Nodes

CREATE Nodes

```
()
```

A node is expressed by parentheses / round brackets and in it's simplest form is just an empty object.

CREATE Nodes

```
()
(:Person)
```

A label gives context to the node. The syntax is colon (`:`), then `Label`.

CREATE Nodes

```
()
(:Person)
(:Person:Actor)
```

A node can be created with multiple labels

CREATE Nodes

```
()
(:Person)
(:Person:Actor)
({name: "John Doe"})
```

Properties make up the content of the node and are put between curly brackets ({}).

CREATE Nodes

```
()
(:Person)
(:Person:Actor)
({name: "John Doe"})
(:Person {name: "John Doe"})
```

You can combine a label and property.

CREATE Nodes

```
()
(:Person)
(:Person:Actor)
({name: "John Doe"})
(:Person {name: "John Doe"})
(:Person:Actor {name: "John Doe", age: 50})
```

You can combine labels and properties.

CREATE Nodes

```
()
(:Person)
(:Person:Actor)
({name: "John Doe"})
(:Person {name: "John Doe"})
(:Person:Actor {name: "John Doe", age: 50})
(john:Person:Actor {name: "John Doe", age: 50})
```

A variable is needed if you want to do something with the node after creating it

Your turn

All code here

https://bit.ly/fundamentals-neo4j-5x

Practice CREATE Nodes

Create an empty node

```
CREATE ();
```

001

Create a node with a label

```
CREATE (:ICanFindIt);
```

002

Create a node with multiple labels

```
CREATE (:ICanFindIt:MoreContext);
```

003

Practice CREATE Nodes

Properties, no labels

```
CREATE ({name: "John Doe"});
```

004

Properties and Labels

```
CREATE (:Person {name: "John Doe"});
```

005

Mulitple labels and properties

```
CREATE (:Person:Actor {name: "Jane Stag", age:40});
```

006

Practice CREATE Nodes

Create and Return

```
CREATE (jeff:Person {name: "Jeff Fawn"})
RETURN jeff;
```

007

MATCH Nodes

```
()
```

Match any (& all) nodes

MATCH Nodes

```
()
(:Person)
```

Pattern matches nodes with a specific label (it filters on the label)

MATCH Nodes

```
()
(:Person)
(:%)
```

Pattern matches nodes with at least one label.

MATCH Nodes

```
()
(:Person)
(:%)
(:!(Person)&(Actor|Actress))
```

Does have an `Actor` or `Actress` label

Doesn't have a `Person` Label

Logical operators can be applied on the labels. NOT (`!`), AND (`&`), OR (`|`)

MATCH Nodes

```
()
(:Person)
(:%)
(:!(Person)&(Actor|Actress))
(x WHERE x.name = "John Doe" AND x.age > 50)
```

Filtering on content (properties) is also possible.

This does require a variable (x in this case).

MATCH Nodes

```
()
(:Person)
(:%)
(:!(Person)&(Actor|Actress))
(x WHERE x.name = "John Doe" AND x.age > 50)
(x:!(Person)&(Actor|Actress) WHERE x.name = "John
Doe")
```

For an efficient pattern match, you should be as precise as possible

Your turn

Practice MATCH Nodes

Count all the nodes

```
MATCH ()
RETURN count(*);
```

008

Count all the nodes with a label

```
MATCH (:%)
RETURN count(*);
```

009

Count all the nodes with a specific label

```
MATCH (:Person)
RETURN count(*);
```

010

Practice MATCH Nodes

Count all the nodes without a label

```
MATCH (:!%)
RETURN count(*);
```

011

Logical operator

```
MATCH (:!(Person)|Actor))
RETURN count(*);
```

012

Practice MATCH Nodes

Find all nodes with a name of ‘John Doe’

```
MATCH (x WHERE x.name = "John Doe")
RETURN count(*);
```

013

Find all the Person nodes with a name of ‘Jane Stag’ and an age > 35

```
MATCH (x:Person WHERE x.name = "Jane Stag" AND x.age > 35)
RETURN count(*);
```

014

Tidying up

Remove all nodes that have no labels, or have ‘ICanFindIt’ as a label

```
MATCH (x:!(%)|ICanFindIt)
DELETE x;
```

Relationships

CREATE Relationships

To create a relationship, the following conditions need to be satisfied:

- It must have one starting node and one ending node
  - *this can be the same node*
- It must have a direction
- It must have a type
  - *it can only have one type*

CREATE Relationships

```
()-[:ACTED_IN]->()
```

This creates two empty nodes and one relationship between them. A relationship uses square brackets / box brackets.

CREATE Relationships

```
()-[:ACTED_IN]->()
(x)-[:ACTED_IN]->(y)
```

Assuming x and y are the variables representing two nodes that were MATCHed earlier, this creates one relationship.

What happens if either x or y are not defined?

CREATE Relationships

```
()-[:ACTED_IN]->()
(x)-[:ACTED_IN]->(y)
(x)-[:ACTED_IN {roles: ["Himself"] } ]->(y)
```

Assuming `x` and `y` are the variables representing two nodes that were `MATCH`ed earlier, this creates one relationship with a `roles` property.

Your turn

Practice CREATE Relationships

Creates a relationship

```
CREATE ()-[:DOESMORETHANMOSTTHINK]->();
```

016

... and 2 nodes

Practice CREATE Relationships

Match nodes first

```
MATCH (source:Person WHERE source.name = "John Doe")
MATCH (target:Movie WHERE target.title = "The Matrix")
CREATE (source)-[:LIKES]->(target);
```

017

Practice CREATE Relationships

Add properties to a relationship

```
MATCH (source:Person WHERE source.name = "Jane Stag")
MATCH (target:Movie WHERE target.title = "The Matrix")
CREATE (source)-[:ACTED_IN {roles: ["Herself"] }]->(target);
```

018

MATCH Relationships

```
()-[]-()
()--()
```

This pattern matches any single hop relationship between any two nodes.

The direction doesn't matter.

MATCH Relationships

```
()-[]-()
()-[]->()
()-->()
()<-[]-()
()<--()
```

This pattern matches any single hop relationship between any two nodes.

The direction does matter.

MATCH Relationships

```
()-[]-()
()-[]->()
()-[:ACTED_IN]->()
```

This pattern matches any single hop relationship between any two nodes filtering for a specific type of relationship.

MATCH Relationships

```
()-[]-()
()-[]->()
()-[:ACTED_IN]->()
()-[:ACTED_IN|DIRECTED]->()
```

While the logical operators will work for relationship types, remember that any given relationship can (and must) only have one type.

This pattern matches any single hop relationship between any two nodes filtering for several specific types of relationship.

MATCH Relationships

```
()-[]-()
()-[]->()
()-[:ACTED_IN]->()
()-[:ACTED_IN|DIRECTED]->()
()-[ai:ACTED_IN WHERE ai.roles = ["Herself"] ]-
>()
```

This pattern matches any single hop relationship between any two nodes filtering for a specific type of relationship and also filtering on the content of the relationship property.

MATCH Relationships

- Relationship Patterns
- Quantified Relationships
- Quantitative Path Patterns
- …

```
()-[*]->()
()-[*1..2]->()
()-[:TYPE*1..2]
```

https://neo4j.com/docs/cypher-manual/current/patterns/reference/

neo4j

Your turn

Practice MATCH Relationships

Count ACTED_IN relationships

```
MATCH ()-[:ACTED_IN]->() RETURN count(*);
```

019

Count ACTED_IN or DIRECTED relationships

```
MATCH ()-[:ACTED_IN|DIRECTED]->() RETURN count(*);
```

020

Practice MATCH Relationships

What is being counted here?

```
MATCH (tom:Person)-[ai:ACTED_IN]->(m:Movie)
WHERE tom.name = "Tom Hanks"
RETURN count(tom);
```

Patterns

A bit of cleanup

Remove relationships first

```
MATCH ()-[r:DOESMORETHANMOSTTHINK|LIKES|ACTED_IN]->()
WHERE r.roles = ["Herself"] OR r.roles IS NULL
DELETE r;
```

022

Then the extra people

```
MATCH (p:Person )
WHERE p.name IN ["John Doe","Jane Stag", "Jeff Fawn"]
DELETE p;
```

023

A bit more cleanup

Delete the empty nodes

```
MATCH (x:!% WHERE isEmpty(properties(x)))
DELETE x;
```

024

Magical Mr White

How many results does this query return?

```
MATCH ()-[x:ACTED_IN WHERE x.roles = ["Mr. White"]]-()
RETURN x;
```

025

How many Mr White?

Recommendations with Cypher

Setting the stage

- Real-time recommendations
  - What you've bought previously
  - What others have bought that is similar to you
  - What's available
  - Similar items
  - etc

Recommending for Tom Hanks

Problem statement

- Recommend good matches to act with Tom Hanks in upcoming movies
- They should not have acted with him before
- The strength of the recommendation is determined by coactors. People that have acted with both Tom Hanks and the recommended person.

Recommendation code here

https://bit.ly/fundamentals-reco-neo4j-5x

Finding Tom

Inline WHERE

```
MATCH (tom:Person WHERE tom.name = "Tom Hanks")
RETURN tom;
```

001

WHERE Clause

```
MATCH (tom:Person)
WHERE tom.name = "Tom Hanks"
RETURN tom;
```

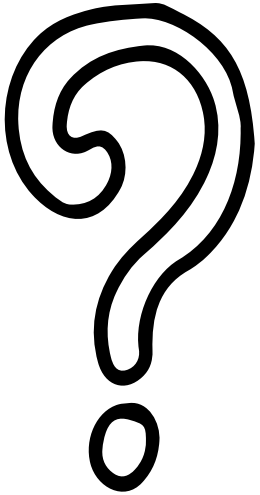
002

Inline

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

003

How can we prove they are the same?



Profile Queries

PROFILE or EXPLAIN

```
PROFILE MATCH (tom:Person WHERE tom.name = "Tom Hanks")
RETURN tom;
```

004

```
PROFILE MATCH (tom:Person)
WHERE tom.name = "Tom Hanks"
RETURN tom;
```

005

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

006

Who acts with Tom?

A "*coactor*" is someone who has acted in the same movie as Tom

```
MATCH (tom:Person)-[:ACTED_IN]->(:Movie)<-[:ACTED_IN]-(coactor:Person)
WHERE tom.name = "Tom Hanks"
RETURN DISTINCT coactor.name;
```

007

Why the DISTINCT ?

Aggregation

How many times have the coactors acted with Tom?

The count is done for every unique coactor name (without the need for a GROUP BY).

```
MATCH (tom:Person)-[:ACTED_IN]->(:Movie)<-[:ACTED_IN]-(coactor:Person)
WHERE tom.name = "Tom Hanks"
RETURN coactor.name, count(*) AS coacts;
```

Who acts with Tom? (continued)

- Who are the most influential coactors?
  - The more films shared, the more influencial

```
MATCH (tom:Person)-[:ACTED_IN]->(:Movie)<-[:ACTED_IN]-(coactor:Person)
WHERE tom.name = "Tom Hanks"
RETURN coactor.name, count(*) AS coacts
ORDER BY coacts DESC;
```

Recommendation

An actor that has acted in the same movie as a coactor (“cocoactor”)

```
MATCH
    (tom:Person)-[:ACTED_IN]->(:Movie)<-[:ACTED_IN]-(coactor:Person),
    (coactor)-[:ACTED_IN]->(:Movie)<-[:ACTED_IN]-(cocoactor:Person)
WHERE
    tom.name = "Tom Hanks"
    AND NOT (tom)-[:ACTED_IN]->(:Movie)<-[:ACTED_IN]-(cocoactor)
    AND tom <> cocoactor
RETURN cocoactor.name, count(coactor) AS strength
ORDER BY strength DESC;
```

010

Recommendation

An actor that has acted in the same movie as a coactor (“cocoactor”)

Don’t recommend actors Tom has already acted with

```
MATCH
    (tom:Person)-[:ACTED_IN]->(:Movie)<-[:ACTED_IN]-(c
    (coactor)-[:ACTED_IN]->(:Movie)<-[:ACTED_IN]-(cocoac
WHERE
    tom.name = "Tom Hanks"
    AND NOT (tom)-[:ACTED_IN]->(:Movie)<-[:ACTED_IN]-(cocoactor)
    AND tom <> cocoactor
RETURN cocoactor.name, count(coactor) AS strength
ORDER BY strength DESC;
```

Recommendation

An actor that has acted in the same movie as a coactor (“cocoactor”)

Don’t recommend Tom!

```
MATCH
    (tom:Person)-[:ACTED_IN]->(:Movie)<-[:ACTED_IN]-(c
    (coactor)-[:ACTED_IN]->(:Movie)<-[:ACTED_IN]-(cocoac
WHERE
    tom.name = "Tom Hanks"
    AND NOT (tom)-[:ACTED_IN]->(:Movie)<-[:ACTED_IN]-(cocoactor)
    AND tom <> cocoactor
RETURN cocoactor.name, count(coactor) AS strength
ORDER BY strength DESC;
```

Extending recommendations