Neo4j is a Graph Database

Contents

About this module

Neo4j enables developers to create applications that are best architected as graph-powered systems that are built upon the rich connectedness of data.

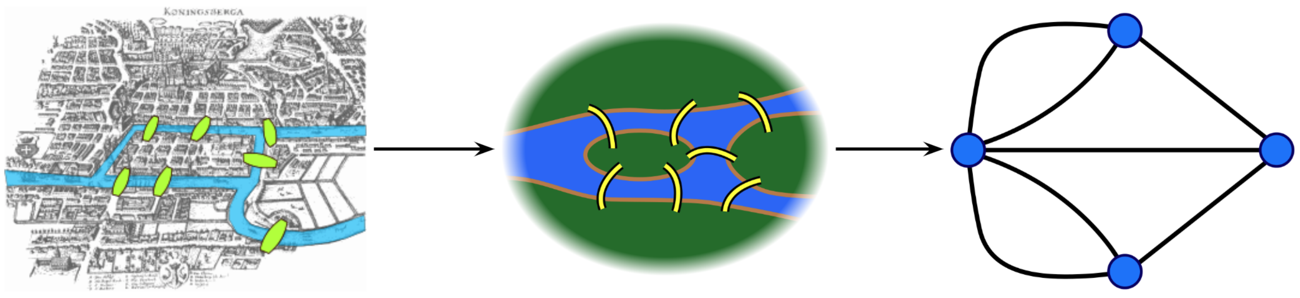
At the end of this module, you will be able to:

- Describe what a graph is.
- Describe what a graph database is.
- Describe some common use cases for using a graph database.
- Describe the Neo4j property graph model.
- Whiteboard a graph data model.

What is a graph?

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

Graphs are a useful way to model data for the purposes of mathematical analysis, and they have been around for quite a long time. The first use of graphs was by Leonhard Euler in 1735; he used graph theory to prove that the famous **Seven Bridges of Konigsberg** problem had no solution.



Konigsberg, Prussia (now Kaliningrad, Russia) is a city spanning the Pregel River. It includes two large islands, which are connected to one another and to the river banks by seven bridges. A map of this arrangement is pictured here.

The Seven Bridges problem is this: Is there any possible route through the city that crosses every bridge exactly once? Some pen-and-paper experimentation will quickly suggest that there is no such route. However, PROVING that there is no solution is much more difficult; Euler invented Graph Theory as a tool to aid this proof. The graph representation of the problem is on the right: four land masses, each represented by a node, and seven bridges, each represented by a relationship.

Euler observed the following:

- The route taken within each landmass is irrelevant. Only the sequence of bridges crossed matters.
- Whenever you enter a land mass via a bridge, you must leave it by a different bridge.
  - ...except at the start (where you leave but do not enter) and at the end (where you enter but do not leave). *Thus, for the problem to be solvable, every land mass (except the start and end) must be touched by an even number of bridges.

Based on the above observations, the problem has no solution, because all four of the land masses are touched by an odd number of bridges.

Anything can be a graph

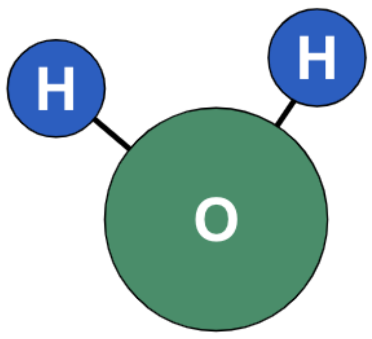
If graphs are an abstract mathematical tool, why do w

Need help? Ask in the Neo4j Community ↗

We care because any dataset can be represented as a graph. Any structure or concept, no matter how simple or complex, can be broken down into a set constituent parts that have some relationship to one another.

For example, the Internet can be modeled as a network of computers.

At the other extreme of complexity, a single molecule can be represented as a graph of its constituent atoms—or an atom represented as a graph of subatomic particles.



And, as Euler showed, there are some clear benefits to representing data in this way. We will discuss many of these as we go. But mainly, graphs are uniquely useful when answering a question that involves following a path along a chain of related items.

You will frequently hear Neo4j people refer to such questions as graph problems—and they are surprisingly common in the real world.

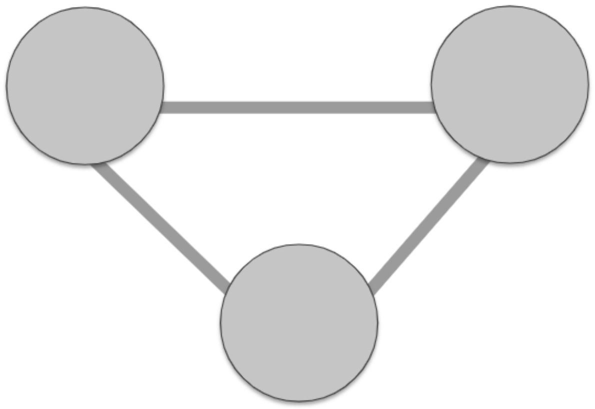
Neo4j’s property graph

The components of A Neo4j graph include:

- Nodes
- Relationships
- Labels
- Properties

Graph terminology and Neo4j’s property graph

- Node = Vertex
- Relationship = Edge

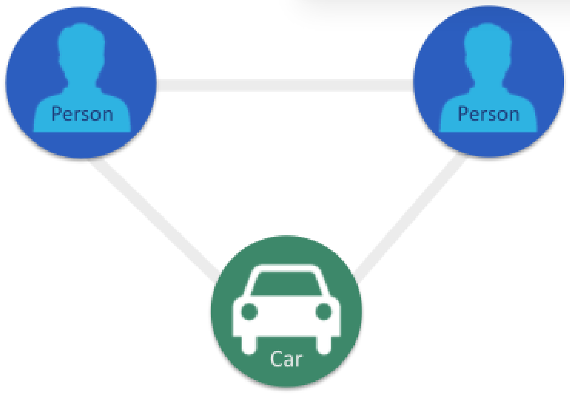


Let’s start by defining the terms vertex and edge. A vertex is an object in a graph. An edge is a connection between two vertices which could be an ordered pair of vertices

Node and Relationship are the Neo4j terms for vertex and edge.

We use these nodes and relationships in Neo4j since we find them more intuitive in representing and communicating - how vertices and edges are used in a property graph.

Nodes



- Represent objects or entities.
- Can be labeled.

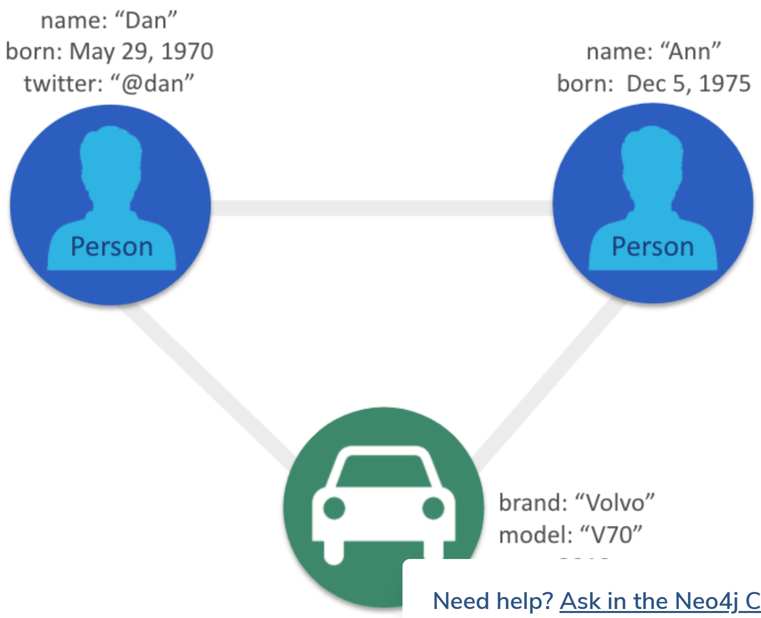
There are two kinds of data that can be associated a node, labels and properties.

Labels provide categorical information. For example, here we have two Person nodes and one Car node.

These labels categorizes these nodes, as a Person or a Car, but they don't provide any specific information about a given node.

Specific information for a node is represented as a properties which you will learn about soon.

Node properties



Need help? Ask in the Neo4j Community ↗

- Represent objects or entities.
- Can be labeled.
- May have properties.

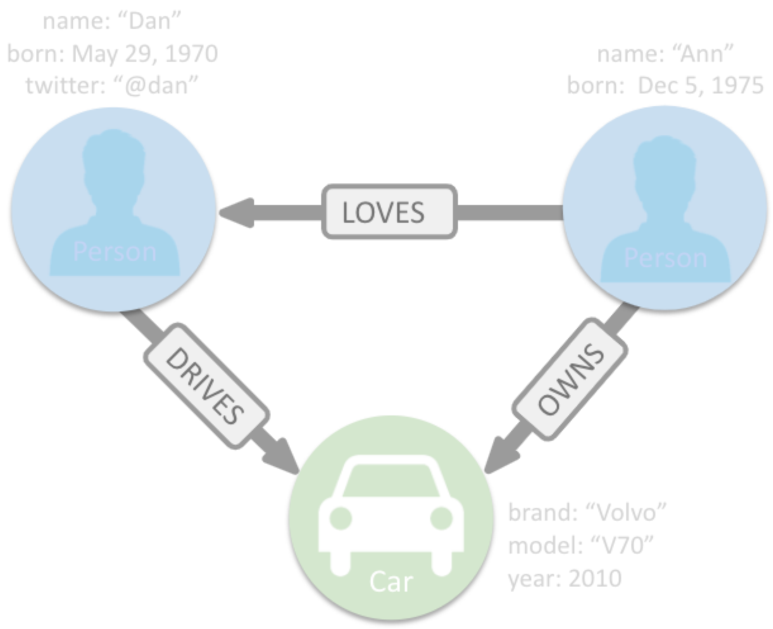
Properties are used to associate specific information with individual nodes.

As shown here, the Person nodes now each have *name* property, One node has a name property with a value, Dan, and the other Person node has a name property with a value, Ann. The name property provides a way to differentiate the nodes. And the Car node has several properties providing details about this particular car.

Nodes do not need to have a label or any properties. However, in practice, a node almost always has at least one label and one or more properties.

This is why we call Neo4j a property graph since it associates data with nodes. As we'll see, having properties in the graph provides value to our application.

Relationships



- Must have a type.
- Must have a direction.

Next, let's talk about relationships. Relationships mus Need help? Ask in the Neo4j Community ↗

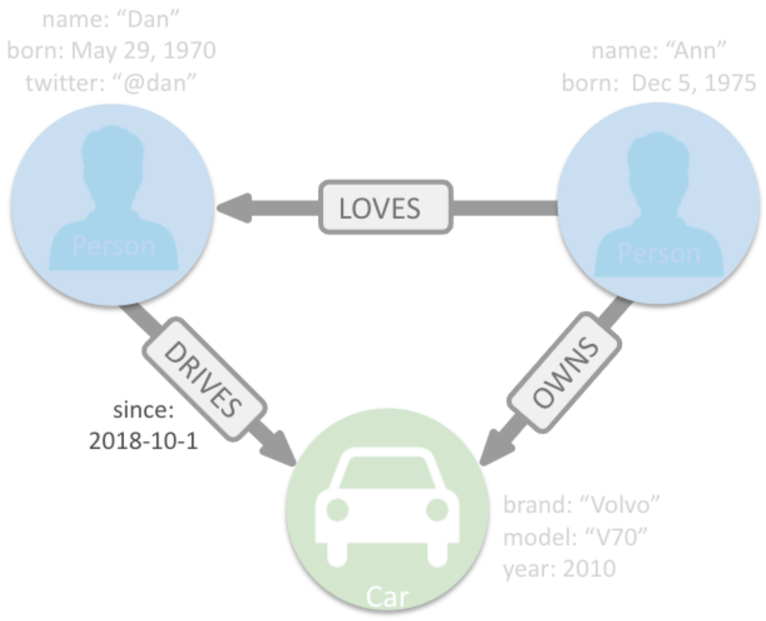
Types are similar to labels since types provide a general category for each relationship.

For example, as we see here, the relationships have the types - DRIVES, OWNS, and LOVES. Each of these types captures how a relationship is used to relate two nodes.

Again, along with a type, each Neo4j relationship must have a direction.

Following the direction shown in this graph we see that "Ann owns a Volvo". If the relationship direction were reversed, it relays a VERY different concept, which is "The Volvo owns Ann."

Relationship properties



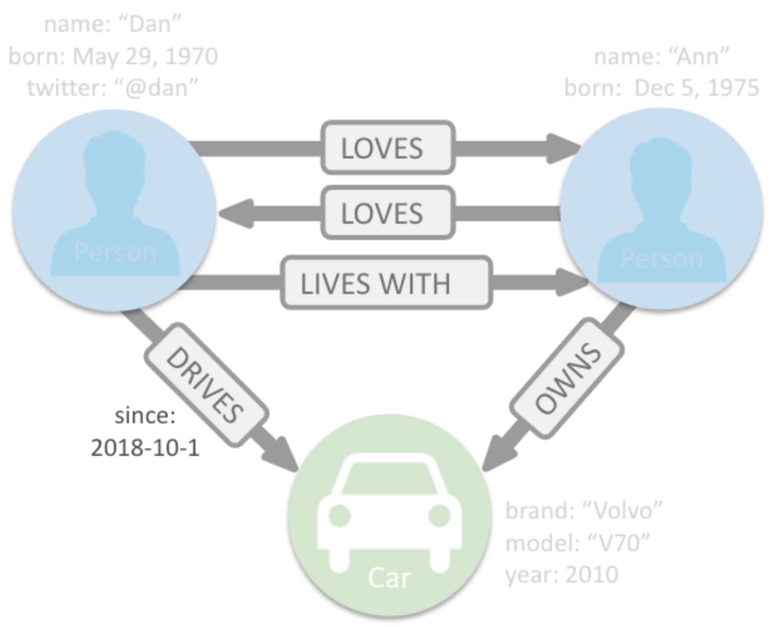
- Must have a type.
- Must have a direction.
- May have properties.

Just like nodes, relationships can have properties that express specific attributes.

Here we see the property 'since' - for the DRIVES relationship. This indicates how long Dan has been driving this particular car.

Multiple relationships

Need help? Ask in the Neo4j Community ↗

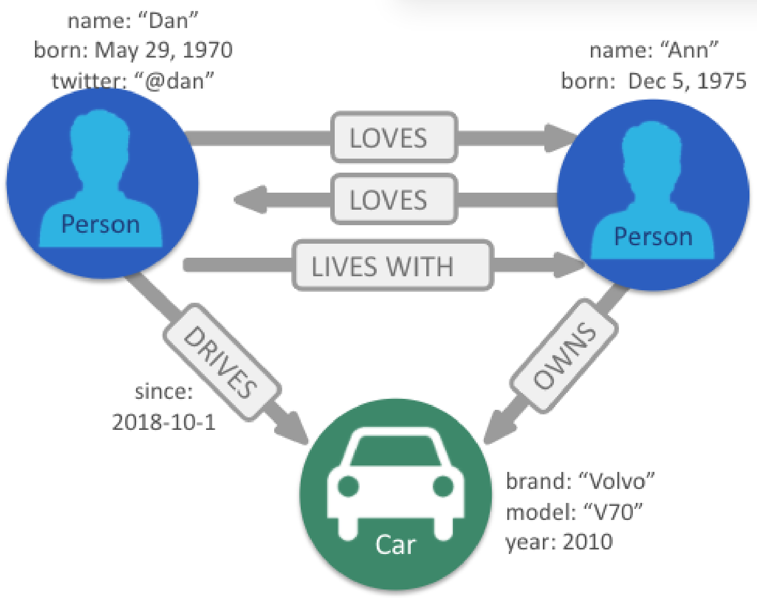


- Must have a type.
- Must have a direction.
- May have properties.
- Nodes can share multiple relationships.

Because relationships are specific, each node can have many relationships with other nodes to fully capture their context.

This is a good thing, otherwise Ann would love Dan, but not vice versa!

A property graph model



- Nodes
  - Represent objects or entities.
  - Can be labeled.
  - May have properties.
- Relationships
  - Must have a type.
  - Must have a direction.
  - May have properties.
- Nodes can share multiple relationships.

Here’s complete, although very simple, property graph for this example.

Mathematically, a node is defined as “a point where two or more edges (relationships) meet.”

This is not at all intuitive! However, it does provide an important insight into how graphs are used.

Since the entire point of modeling data as a graph is to traverse a chain of linked nodes. One useful way of thinking about nodes is that they are waypoints along the traversal route. They contain the information needed to decide which links are good ones to follow, and which can be ignored. Relationships are those links.

The support for properties on both nodes and relationships is what makes Neo4j a property graph. This provides a flexible way of defining varying amounts of data.

Traversing a graph

A very important concept within graph theory is traversal. An important feature of graphs is to follow a chain of nodes and relationships.

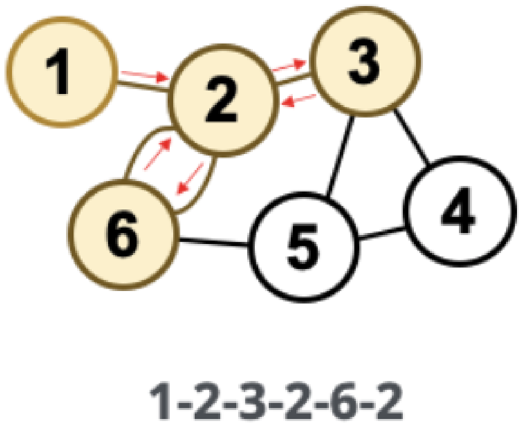
The process of finding such routes is the definition of traversal.

We'll be looking at 3 different ways that traversal can be done:

- Walk
- Trail
- Path

Traversing a graph: Walk

A walk is an ordered, alternating sequence of nodes and relationships where the nodes and relationships can be repeated.



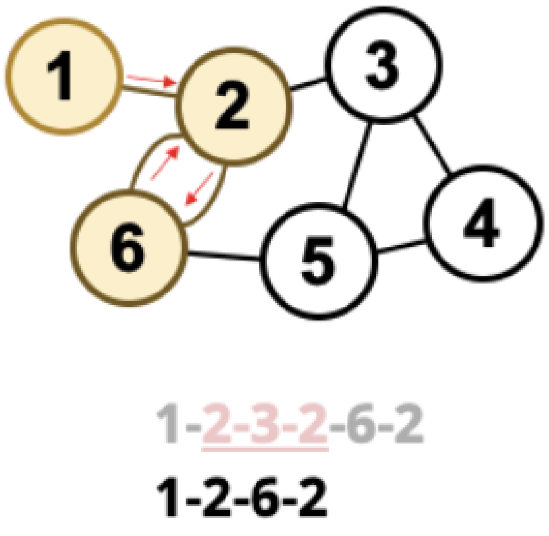
A walk is the most basic type of traversal. A walk is an ordered alternating list of nodes and relationships. For example, in the graph shown here, we could define a walk starting at node 1, hopping to 2, then 3, back to 2, on to 6, and back to 2. There are no restrictions here.

You can traverse relationships multiple times and visit

Need help? Ask in the Neo4j Community ↗

Traversing a graph: Trail

A trail is a walk where no relationships are repeated, but nodes can be visited more than once.



For example, remember the Seven Bridges problem? It uses a trail. You can visit each landmass as many times as you'd like, but you can only cross each bridge once.

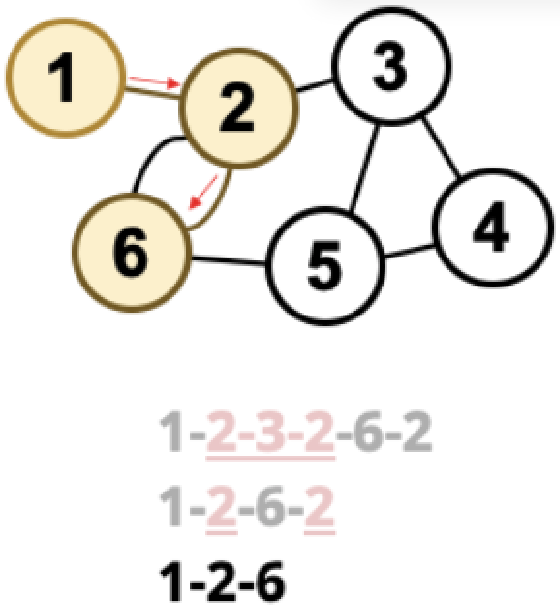
The walk we just performed (1-2-3-2-6-2) is not a valid trail, because we traversed the relationship between 2 and 3 twice.

Instead, a valid trail, as shown here, could start at 1, go to 2, 6, and back to 2. This is permitted since there are two relationships between 6 and 2, and we use each of them once.

Traversing a graph: Path

A path is a trail where no nodes are repeated. That is, all items are unique.

Need help? Ask in the Neo4j Community ↗



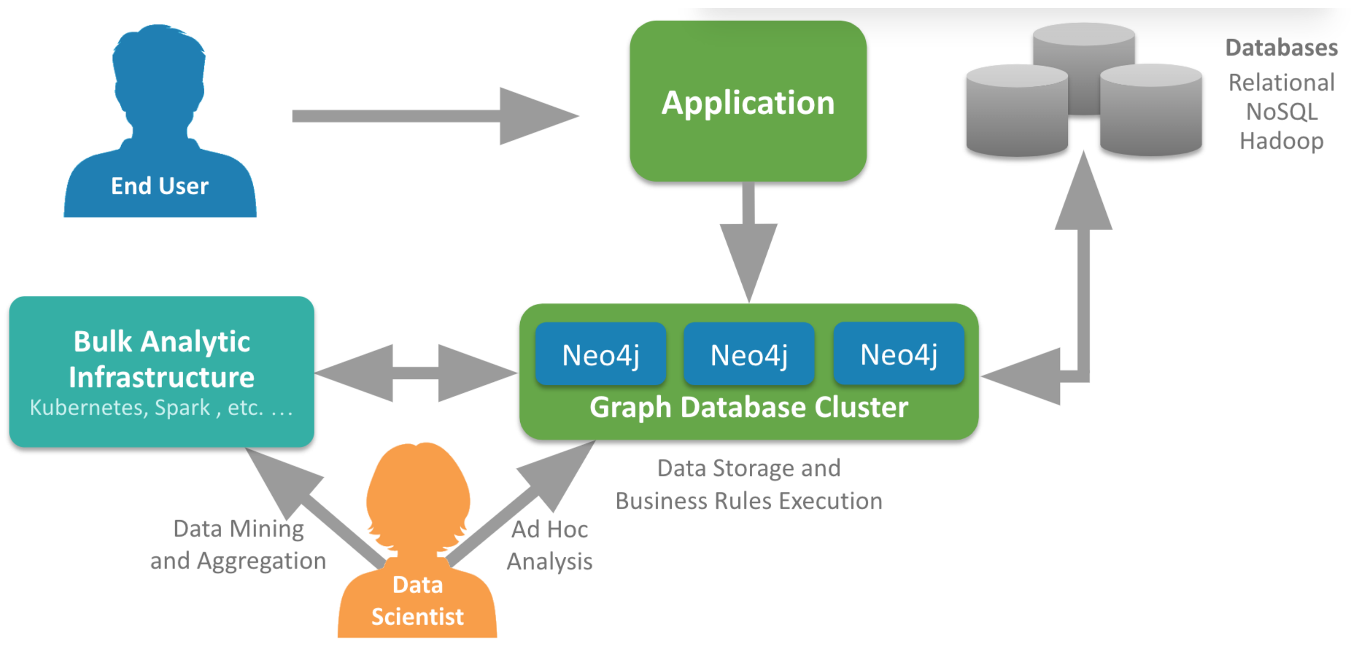
The trail we just followed (1-2-6-2) is not a valid path because it visits node 2 twice. By dropping that last hop, it becomes a valid path.

The problems we solve with Neo4j are almost exclusively done by traversing trails and paths. Each of these terms mean something very specific in the context of graphs.

For example, there are several special kinds of paths that are extremely useful for solving certain problems. Shortest path (the path containing the fewest elements) is the most common of these. A couple of more esoteric examples include:

- Longest path, Hamiltonian path (visits every node in the graph)
- Cycle ("path" that ends at the same node it started on)

What is a graph database?


End User
Application
Databases
Relational
NoSQL
Hadoop
Bulk Analytic Infrastructure
Kubernetes, Spark , etc. ...
Neo4j
Neo4j
Neo4j
Graph Database Cluster
Data Storage and Business Rules Execution
Data Mining and Aggregation
Ad Hoc Analysis
Data Scientist

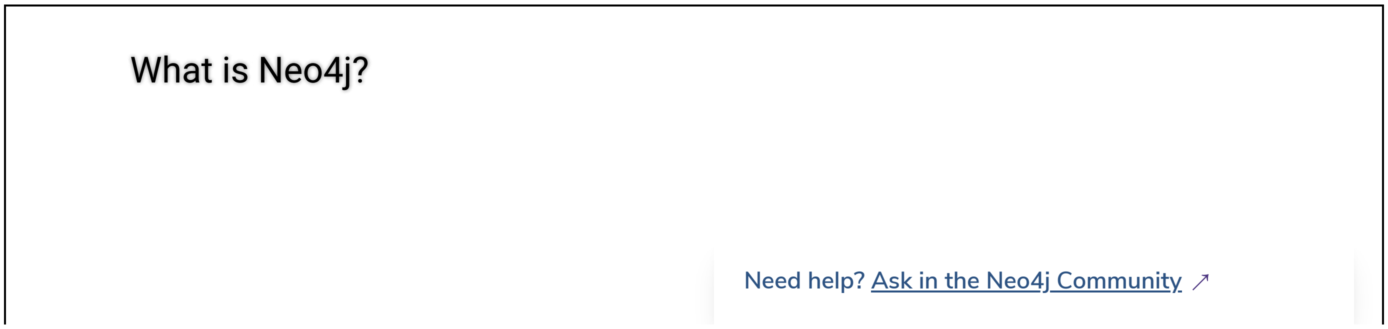
A graph database is an online database management system with Create, Read, Update, and Delete (CRUD) operations working on a graph data model.

Graph databases are often used as part of an Enterprise online transaction processing (OLTP) system. Accordingly, they are normally optimized for transactional performance, and engineered with transactional integrity and operational availability in mind.

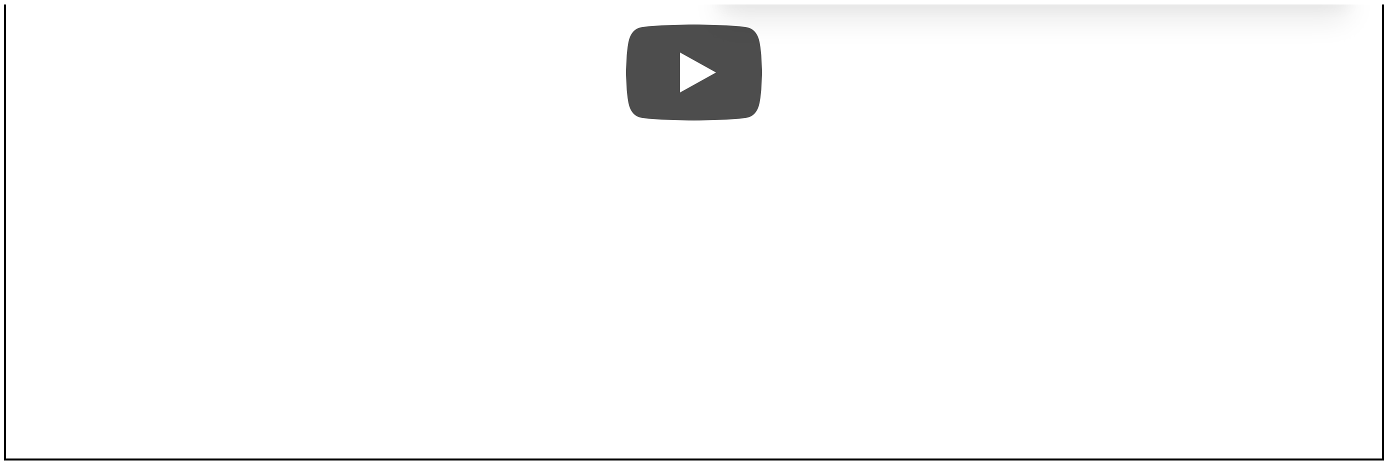
Unlike other databases, relationships take first priority in graph databases. This means your application doesn't have to infer data connections using foreign keys or out-of-band processing, such as MapReduce. By assembling the simple abstractions of nodes and relationships into connected structures, graph databases enable us to build sophisticated models that map closely to our problem domain. Data scientists can use the relationships in a data model to help enterprises make important business decisions.

Neo4j is a native graph database

In this video, you will see the benefits of using the native graph database, Neo4j.



Need help? Ask in the Neo4j Community ↗



Querying using Cypher is intuitive

If you have ever tried to write a SQL statement with a large number of JOINs, you know that you quickly lose sight of what the query actually does, due to the number of tables that are required to connect the data.

SQL statement - listing the employees in the IT Department:

SQL

```sql
SELECT name FROM Person
LEFT JOIN Person_Department
  ON Person.Id = Person_Department.PersonId
LEFT JOIN Department
  ON Department.Id = Person_Department.DepartmentId
WHERE Department.name = "IT Department"
```

Cypher statement - clean and focused since queries are expressed visually:

Cypher

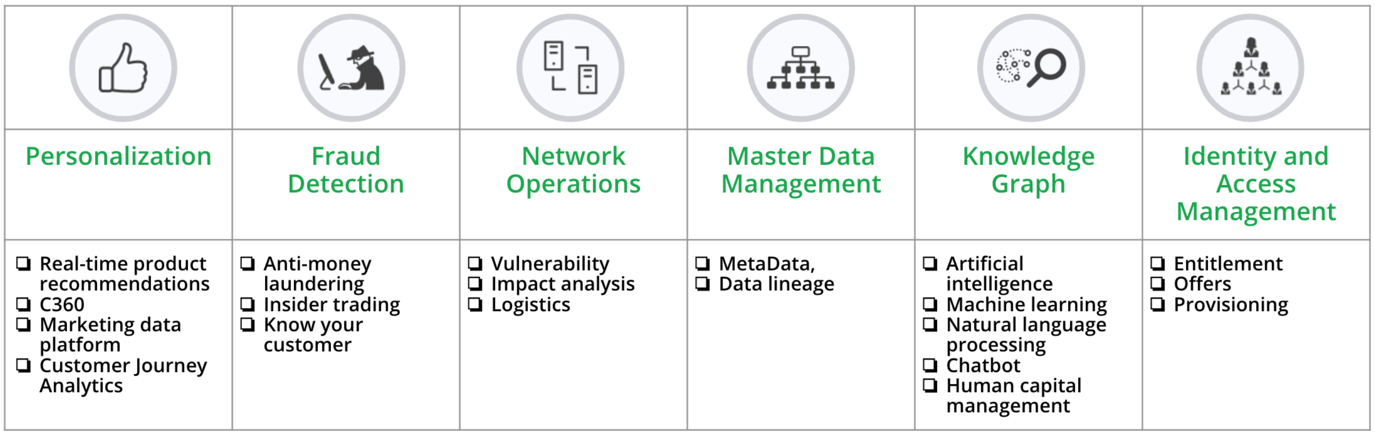
```cypher
MATCH (p:Person)<-[:EMPLOYEE]-(d:Department)
WHERE d.name = "IT Department"
RETURN p.name
```

Use cases for graph databases

The biggest value that graphs bring to an application is their ability to store relationships and connections as first-class entities.

For instance, the early adopters of graph technology built their businesses around the value of data relationships. These companies have now become industry leaders: LinkedIn, Google, Facebook, and PayPal.

There are many uses for graph databases in an enterprise. Here are some of them:



Personalization	Fraud Detection	Network Operations	Master Data Management	Knowledge Graph	Identity and Access Management
❏ Real-time product recommendations ❏ C360 ❏ Marketing data platform ❏ Customer Journey Analytics	❏ Anti-money laundering ❏ Insider trading ❏ Know your customer	❏ Vulnerability ❏ Impact analysis ❏ Logistics	❏ MetaData, ❏ Data lineage	❏ Artificial intelligence ❏ Machine learning ❏ Natural language processing ❏ Chatbot ❏ Human capital management	❏ Entitlement ❏ Offers ❏ Provisioning

You can read more about how customers are using Neo4j here ↗ .

Training data model for this course

For this training, we will use a Movies graph to get you started with accessing the data in a Neo4j database.

The data model is very simple, it includes entities such as:

- Movie
- Person/Actor/Director

that are implemented as nodes in the graph.

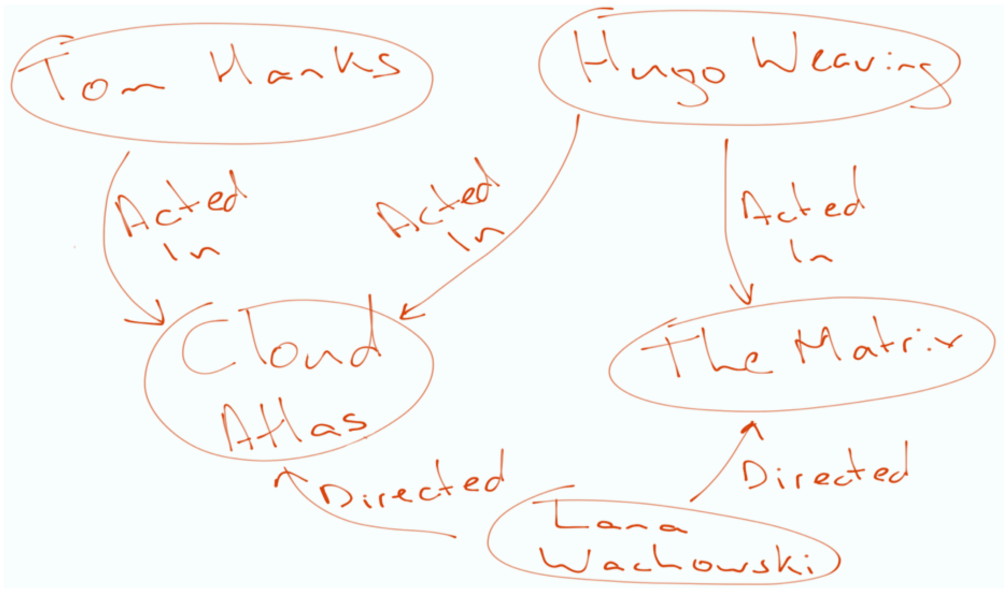
People are related to Movies using these relationships:

- DIRECTED
- ACTED_IN
- WROTE
- REVIEWED

People, who have reviewed movies, are related to each other using the FOLLOWS relationships.

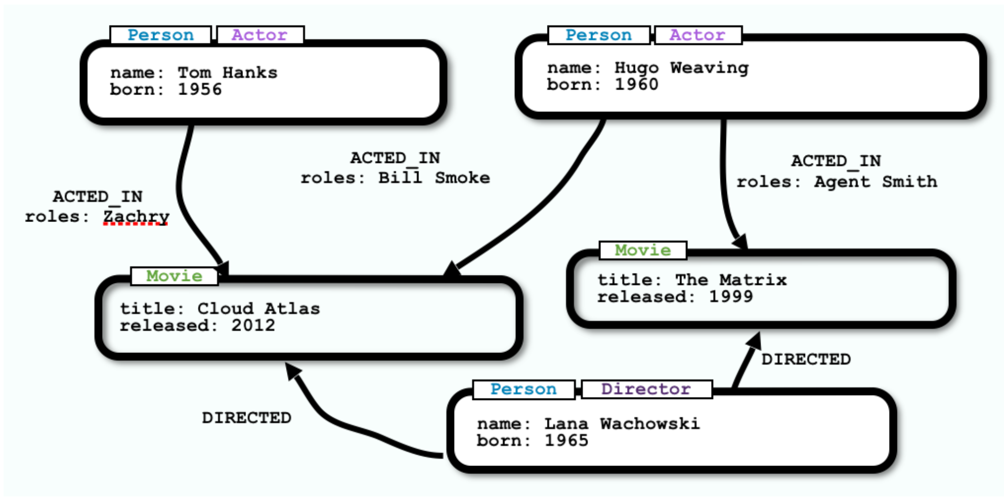
Initial whiteboard the data model

You can use a whiteboard to better understand the graph data model. Stakeholders for an application typically communicate their ideas about how the data will be modeled by drawing bubbles for the entities and directed arrows for the relationships between the entities.



Here we see that people such as Tom Hanks and Hugo Weaving have acted in the movie, Cloud Atlas.

Implementation of the graph data model



Once the graph data model has been agreed-upon, developers then create the nodes with the relationships. When nodes and relationships are added to the graph, additional properties are set for

Need help? Ask in the Neo4j Community ↗

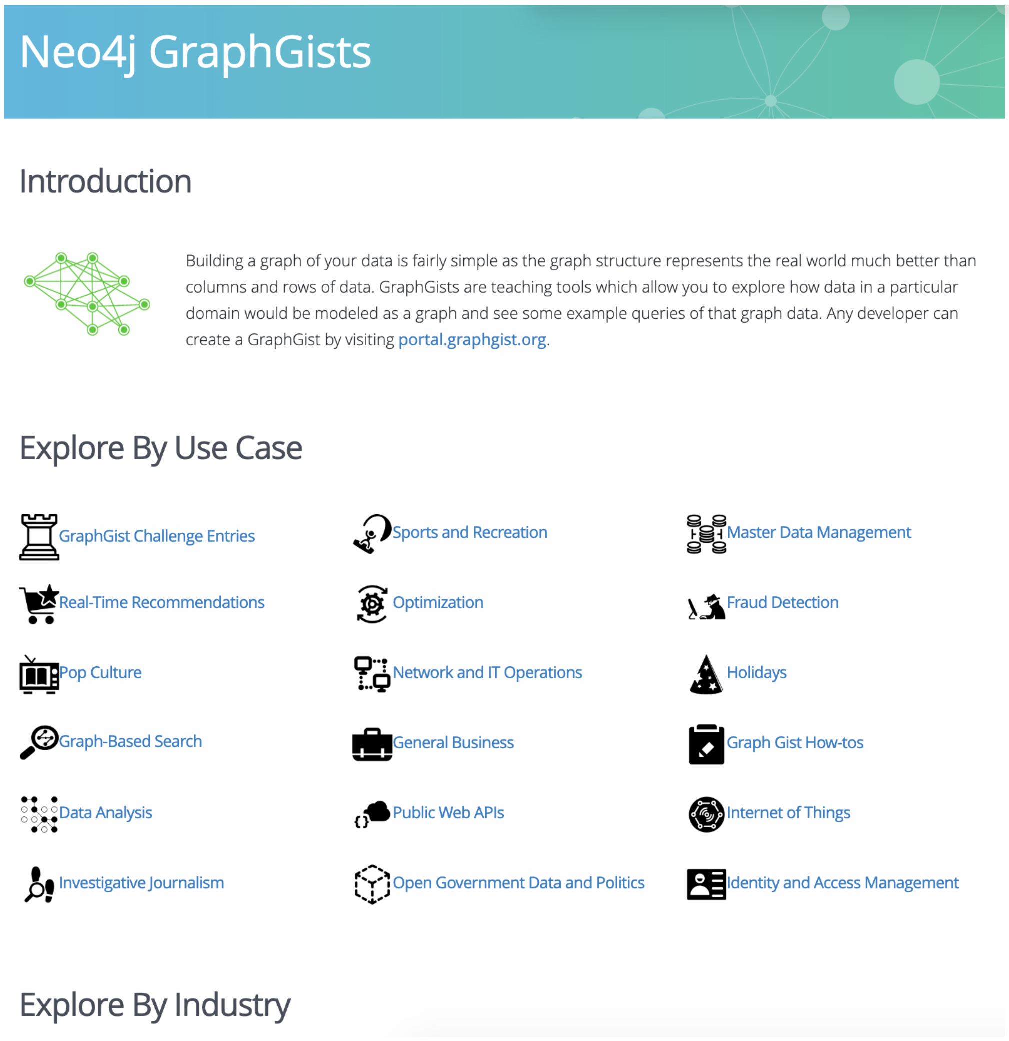
nodes and relationships using the property graph modeling capabilities of Neo4j. Some properties for nodes must be unique such as the name of a person or the title of a movie.

Here is the partial implementation of this graph data model. Notice that some nodes not only have a Person label, but also have a Actor or Director label on them. A Person node has a name property and some Person nodes have a born property. A Movie node has title and released properties. The ACTED_IN relationship has a roles property.

As you learn Cypher in this course, you will explore all of these node and relationship types in the graph.

Exploring graphgists

As a developer just learning Neo4j and Cypher, you will use the very simple Movies graph data model. If, however, you are curious about how other developers have created graph data models for Neo4j, you can explore the graphgists page ↗ .


Neo4j GraphGists

Introduction

Building a graph of your data is fairly simple as the graph structure represents the real world much better than columns and rows of data. GraphGists are teaching tools which allow you to explore how data in a particular domain would be modeled as a graph and see some example queries of that graph data. Any developer can create a GraphGist by visiting portal.graphgist.org.

Explore By Use Case

GraphGist Challenge Entries	Sports and Recreation	Master Data Management
Real-Time Recommendations	Optimization	Fraud Detection
Pop Culture	Network and IT Operations	Holidays
Graph-Based Search	General Business	Graph Gist How-tos
Data Analysis	Public Web APIs	Internet of Things
Investigative Journalism	Open Government Data and Politics	Identity and Access Management

Explore By Industry

Graphgists are developed by a wide range of developers from Neo4j and from the Neo4j user community. You can explore this site to see how other applications have modeled their data and get an appreciation for what can be done in Neo4j.

Example graphgist

For example, here is the *Exploring a Conference* graphgist that has a description of the model, how to load the sample data, and how to query the data.

Need help? Ask in the Neo4j Community ↗

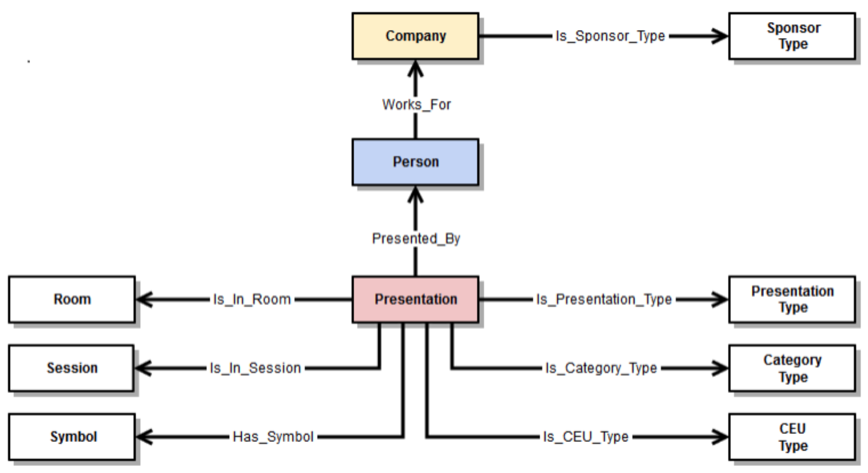
Exploring A Conference

by Dustin Cote

Going to a Conference can be a lot of fun, an important networking opportunity, and a great learning experience. But it can also be hard to choose which workshops to attend. Conference workbooks can help you choose by providing long descriptions of the presentations offered and the biographies of the presenters. But they also can only provide one linear way to look at your options: by date & time. Neo4j and Graph Databases allow you to easily explore the conference by different category types, presenters, companies, vendors, room numbers, and more.

In our example, we will input data from a specific conference: The Annual MOSES Organic Gardening Conference from 2015. This is the largest event in the U.S. about organic & sustainable farming. It has 160 presenters, 281 companies, and 135 workshops & presentations. These were spread out over the course of 3 days, with 16 rooms, and 45 sessions.

Here is our Domain Model:



Example Data: Load & View

Here is one example of a MOSES Conference Presentation: "Does It Pay to Irrigate Pasture Here?" This one happens to have 2 presenters. One of the presenters works for 2 different companies. We also load the room, session time, category & presentation types of this Presentation.

Query 1

```
CREATE (per154:Person {id:154, name:'Tom Kriegl', city:'', state:'', email1:'tskriegl
(per155:Person {id:155, name:'Paul Onan', city:'Amherst Junction', state:'WI', email1
(pres19:Presentation {id:19, name:'Does it Pay to Irrigate Pasture Here?', descriptio
(co34:Company {id:34, booth:'', exhibitor:'FALS
```

Need help? Ask in the Neo4j Community ↗

As you gain experience with Cypher and think about modeling your application, check out some of these graphgists for ideas for how to model your data. We recommend that you take our training course, *Graph Data Modeling for Neo4j*, which introduces you to the best practices for modeling data and implementing graph data models in Neo4j.

Check your understanding

Question 1

What elements make up a graph?

Select the correct answers.

- [ ] tuples
- [✓ nodes]
- [ ] documents
- [✓ relationships]

Question 2

From a data scientist perspective, which element is key for analyzing the data in the graph?

Select the correct answer.

- [ ] node
- [ ] property
- [✓ relationship]
- [ ] join

Question 3

In Neo4j, you can attach properties to the following elements in the graph?

Select the correct answers.

- [ ] label
- [x] node
- [x] relationship
- [ ] direction

Summary

You can now:

- Describe what a graph is.
- Describe what a graph database is.
- Describe some common use cases for using a graph database.
- Describe the Neo4j property graph model.
- Whiteboard a graph data model.

Prev
‹ Overview of Neo4j 4.x

Next
The Neo4j Graph Platform ›

Thank you for your feedback!

We will use this information to help personalise your experience.

Contents

Need help? Ask in the Neo4j Community ↗

Contact Us →

US: 1-855-636-4532
Sweden +46 171 480 113
UK: +44 20 3868 3223
France: +33 (0) 8 05 08 03 44

Learn

- Sandbox
- Neo4j Community Site
- Neo4j Developer Blog
- Neo4j Videos
- GraphAcademy
- Neo4j Labs

Social

- Twitter
- Meetups
- Github
- Stack Overflow

Want to Speak? Get $ back.

Need help? Ask in the Neo4j Community ↗