Introduction to Graph Databases

Table of Contents

About this module . . . 1
- The evolution of graph databases . . . 1
- What Is a graph database? . . . 1
- The case for graph databases . . . 1
- What is a graph? . . . 2
- Modeling relational to graph . . . 4
  - Run-time behavior: RDBMS vs graph . . . 5
  - How we model: RDBMS vs graph . . . 5
- How does Neo4j support the property graph model? . . . 7

Check your understanding . . . 8
- Question 1 . . . 8
- Question 2 . . . 8
- Question 3 . . . 8
- Question 4 . . . 8

Summary . . . 10

Grade Quiz and Continue . . . 11

About this module

The Neo4j Graph Platform 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 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.

The evolution of graph databases

Today's business and user requirements demand applications that connect more and more of the world's data, yet still expect high levels of performance and data reliability. Many applications of the future will be built using graph databases like Neo4j.

In this video, you will learn how the need for graph databases has evolved.

https://youtu.be/5Tl8WcaqZoc

What Is a graph database?

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 generally built for use with online transaction processing (OLTP) systems. 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.

The case for graph databases

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

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

As pioneers in graph technology, each of these enterprises had to build their own graph database from scratch. Fortunately for today's developers, that's no longer the case, as graph database

technology is now available off the shelf.

In this video, you will learn how graph databases help you to model real-world data that needs to be connected as well as how Neo4j is used to solve real problems facing enterprises today.

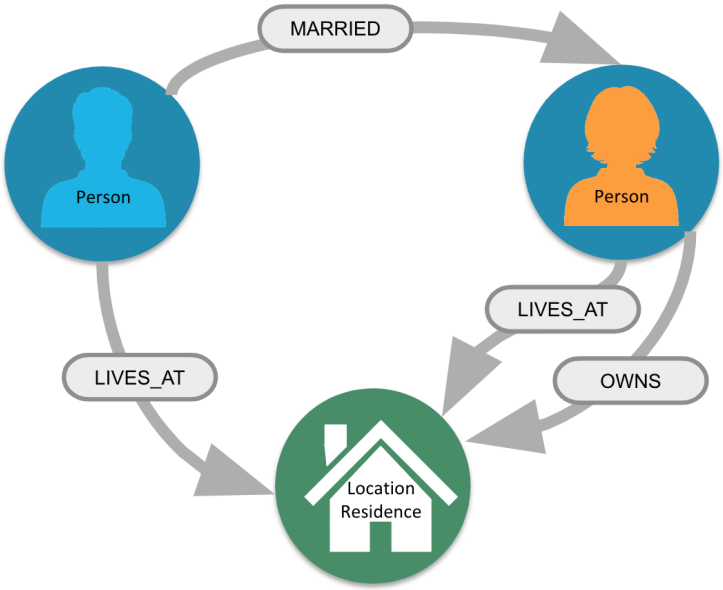
https://youtu.be/-dCeFEqDkUI

What is a graph?

A graph is composed of two elements: **nodes** and **relationships**.

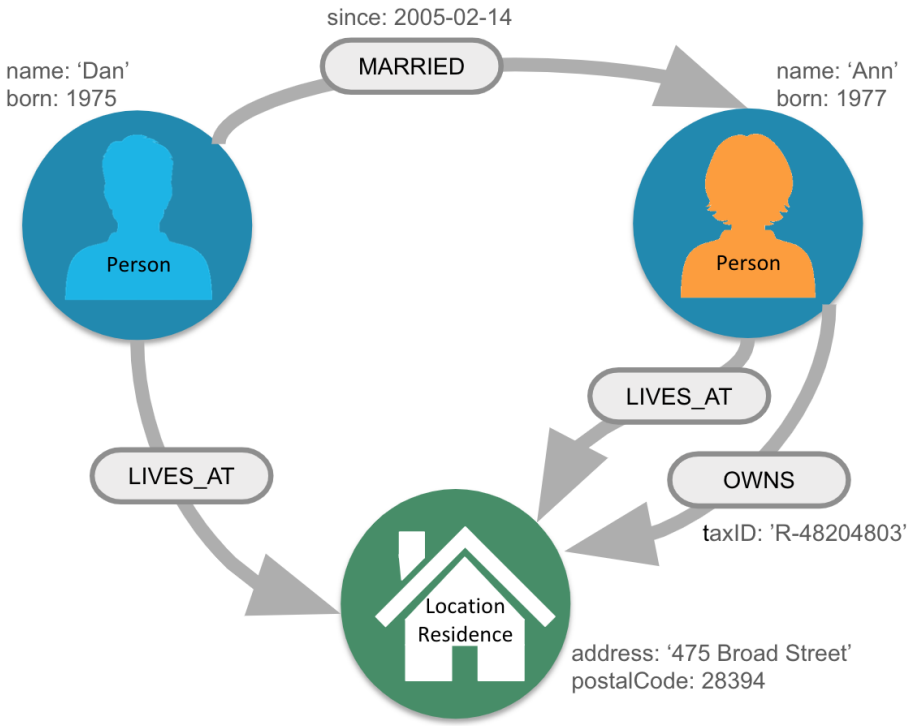
Each node represents an entity (a person, place, thing, category or other piece of data). With Neo4j, nodes can have **labels** that are used to define types for nodes. For example, a *Location* node is a node with the label *Location*. That same node can also have a label, *Residence*. Another *Location* node can also have a label, *Business*. A label can be used to group nodes of the same type. For example, you may want to retrieve all of the *Business* nodes.

Each relationship represents how two nodes are connected. For example, the two nodes *Person* and *Location*, might have the relationship *LIVES_AT* pointing from a *Person* node to *Location* node. A relationship represents the verb or action between two entities. The *MARRIED* relationship is defined from one *Person* node to another *Person* node. Although the relationship is defined as directional, it can be queried in a non-directional manner. That is, you can query if two *Person* nodes have a *MARRIED* relationship, regardless of the direction of the relationship. For some data models, the direction of the relationship is significant. For example, in Facebook, using the *KNOWS* relationship is used to indicate which *Person* invited the other *Person* to be a friend.



This general-purpose structure allows you to model all kinds of scenarios: from a system of roads, to a network of devices, to a population's medical history, or anything else defined by relationships.

The Neo4j database is a property graph. You can add **properties** to nodes and relationships to further enrich the graph model.



This enables you to closely align data and connections in the graph to your real-world application. For example, a *Person* node might have a property, *name* and a *Location* node might have a property, *address*. In addition, a relationship, *MARRIED* ,might have a property, *since*.

In this video, you will learn how to model property graphs containing nodes and relationships and how Cypher is used to access a graph database.

https://youtu.be/NH6WoJHN4UA

Modeling relational to graph

Many applications' data is modeled as relational data. There are some similarities between a relational model and a graph model:

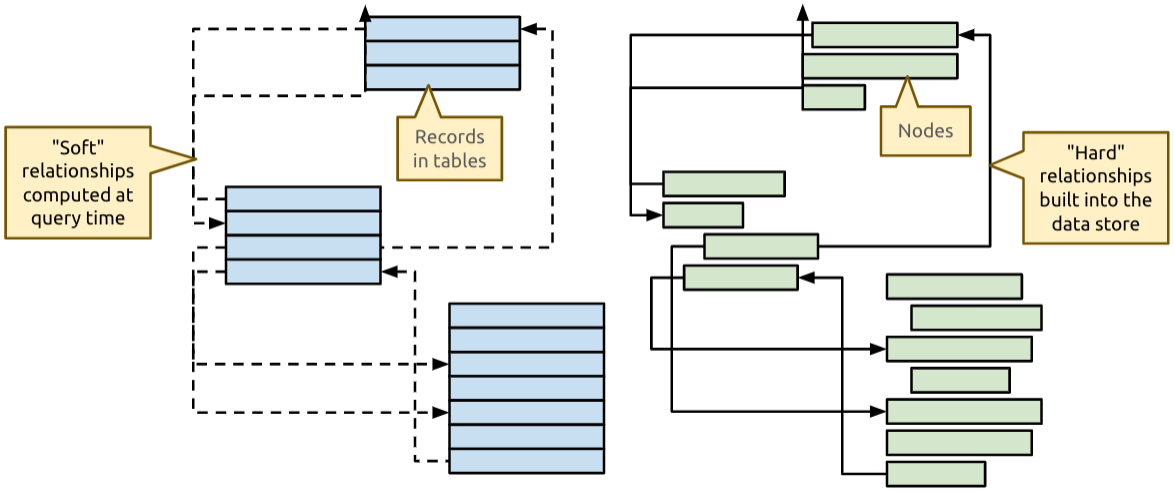
Relational	**Graph**
Rows	Nodes
Joins	Relationships
Table names	Labels
Columns	Properties

But, there are some ways in which the relational model differs from the graph model:

Relational	Graph
Each column must have a field value.	Nodes with the same label aren't required to have the same set of properties.
Joins are calculated at query time.	Relationships are stored on disk when they are created.
A row can belong to one table.	A node can have many labels.

Run-time behavior: RDBMS vs graph

How data is retrieved is very different between an RDBMS and a graph database:



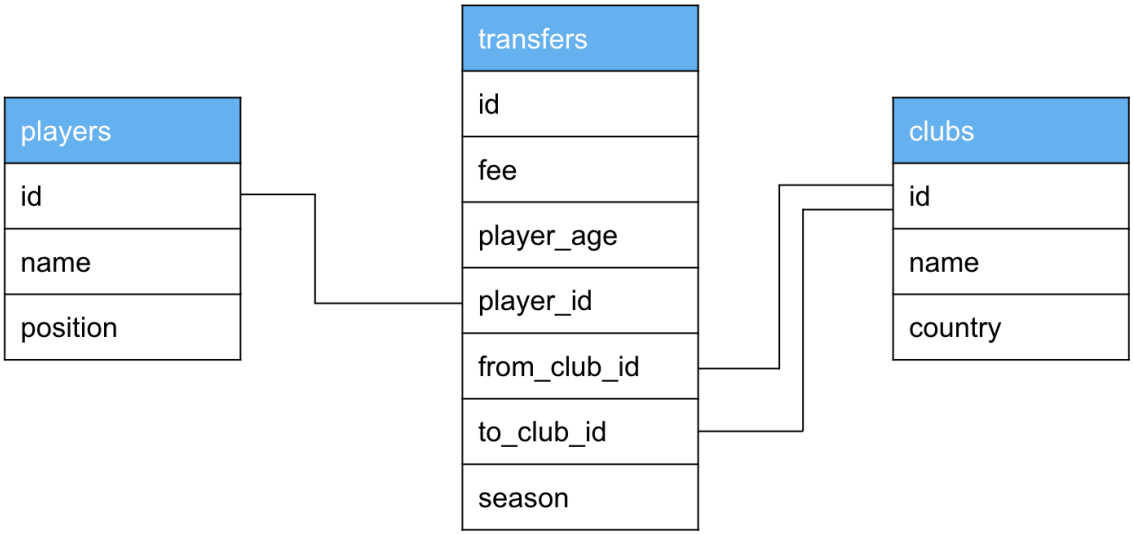
How we model: RDBMS vs graph

How you model data from relational vs graph differs:

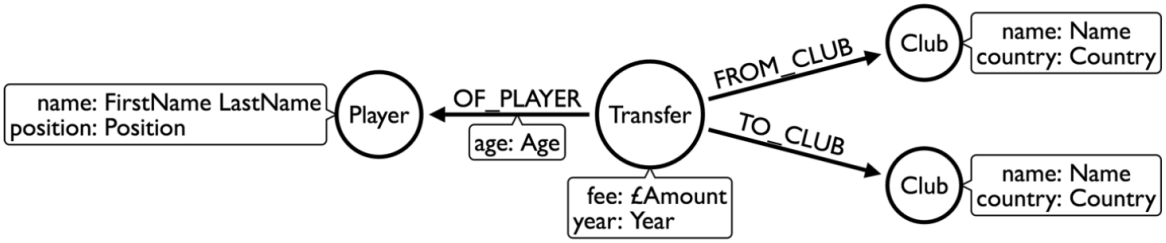
Relational	Graph
Try and get the schema defined and then make minimal changes to it after that.	It's common for the schema to evolve with the application.
More abstract focus when modeling i.e. focus on classes rather than objects.	Common to use actual data items when modeling.

If we were modeling a football transfers graph in relational and graph databases these diagrams show what common approaches might look like.

Here is the relational model:

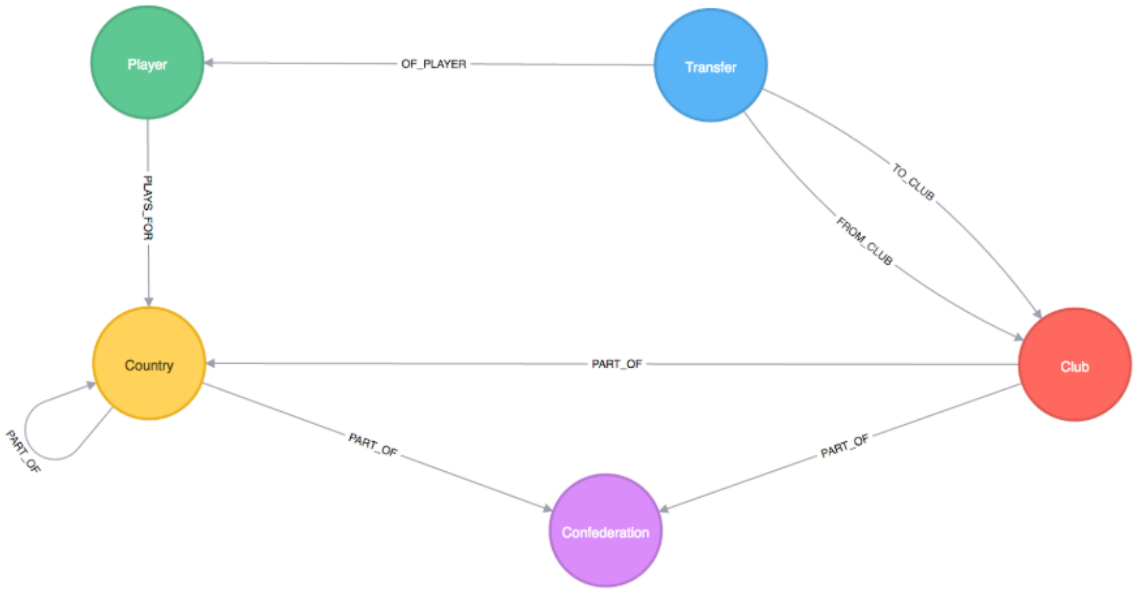


and here is the corresponding graph model:



With the graph model we might sketch out examples with actual values and derive the 'schema' while doing that modeling process.

In Neo4j, the data model might evolve to something like this:



How does Neo4j support the property graph model?

- Neo4j is a **Database** - use it to reliably **store information** and **find it later**.
- Neo4j's data model is a **Graph**, in particular a **Property Graph**.
- **Cypher** is Neo4j's graph query language (**SQL for graphs!**).
- Cypher is a declarative query language: it describes **what** you are interested in, not **how** it is acquired.
- Cypher is meant to be very **readable** and **expressive**.

Check your understanding

Question 1

What elements make up a graph?

Select the correct answers.

- ▢ tuples
- ▢ nodes
- ▢ documents
- ▢ relationships

Question 2

Suppose that you want to create a graph to model customers, products, what products a customer buys, and what products a customer rated. You have created nodes in the graph to represent the customers and products. In this graph, what relationships would you define?

Select the correct answers.

- ▢ BOUGHT
- ▢ IS_A_CUSTOMER
- ▢ IS_A_PRODUCT
- ▢ RATED

Question 3

What query language is used with a Neo4j Database?

Select the correct answer.

- ▢ SQL
- ▢ CQL
- ▢ Cypher
- ▢ OPath

Question 4

Why is Lisp used for artificial intelligence?

Select the correct answer.

- ▢ It is an excellent prototyping tool and good at tackling problems

- ▢ LISP is powerful. The code or data distinction is weaker, so it feels more extensible than other programming languages which make it feels like a domain specific language
- ▢ LISP is the best programming answer on earth

Summary

You should now 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.

Grade Quiz and Continue