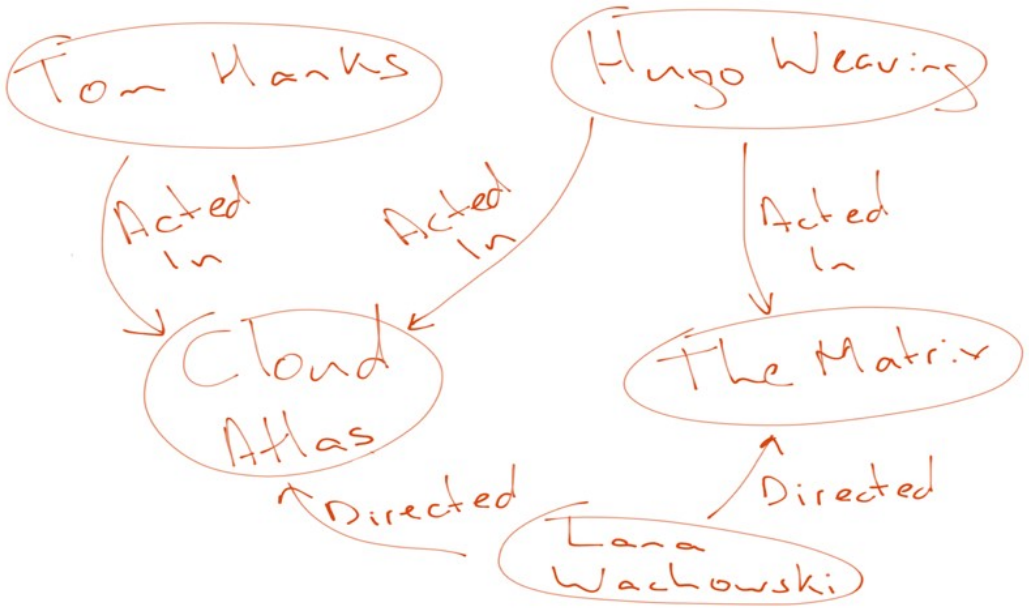


Graph Modeling Guidelines

neo4j.com

7 mins read

Goals This guide is designed to walk you through the graph data modeling lifecycle of Neo4j. You will be introduced to the basic process of designing a graph data model that can answer a wide range of business questions across... Read more →

Introduction

If you have ever worked with an object model or an entity-relationship diagram, the labeled property graph model will seem familiar.

Graph data modeling is the process in which a user describes an arbitrary domain as a connected graph of nodes and relationships with properties and labels. A Neo4j graph data model is designed to answer questions in the form of Cypher queries and solve business and technical problems by organizing a data structure for the graph database.

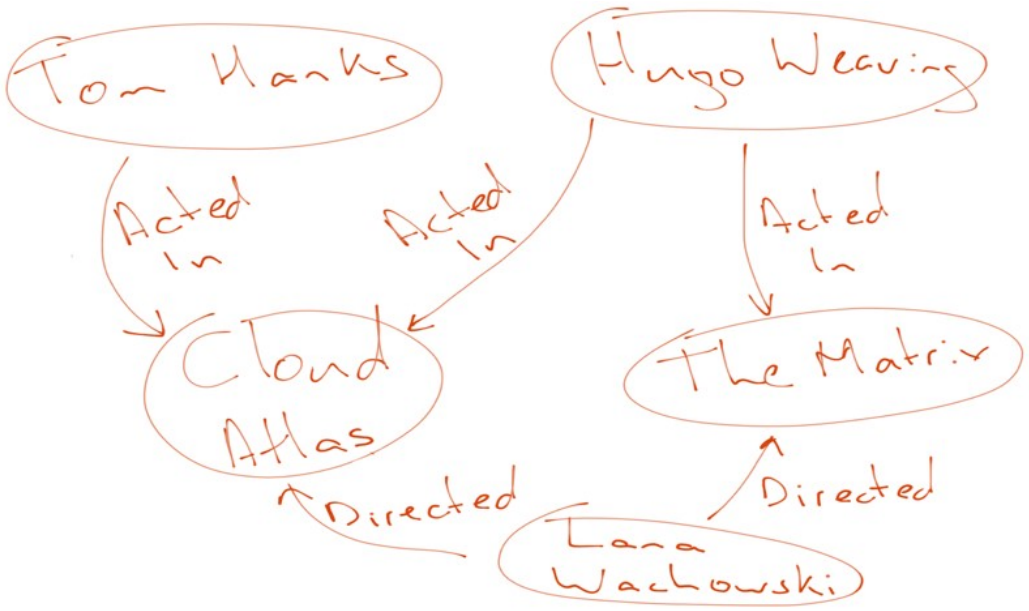
Graph Data Model = Whiteboard-Friendly

The graph data model is often referred to as being “whiteboard-friendly”. Typically, when designing a data model, people draw example data on the whiteboard and connect it to other data drawn to show how different items connect. The whiteboard model is then re-formatted and structured to fit normalized tables for a relational model.

A similar process exists in graph data modeling, as well. However, instead of modifying the data model to fit a normalized table structure, the graph data model stays exactly as it was drawn on the whiteboard. This is where the graph data model gets its name for being “whiteboard-friendly”.

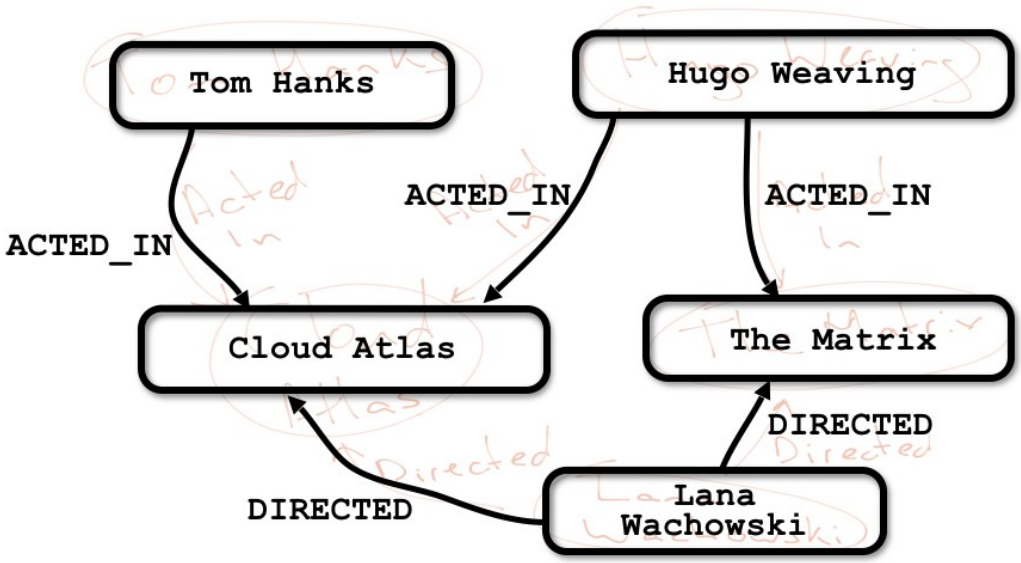
Let us look at an example to demonstrate this. In the whiteboard drawing below, we have a data set about the movie “The Matrix”.

Matrix – whiteboard model



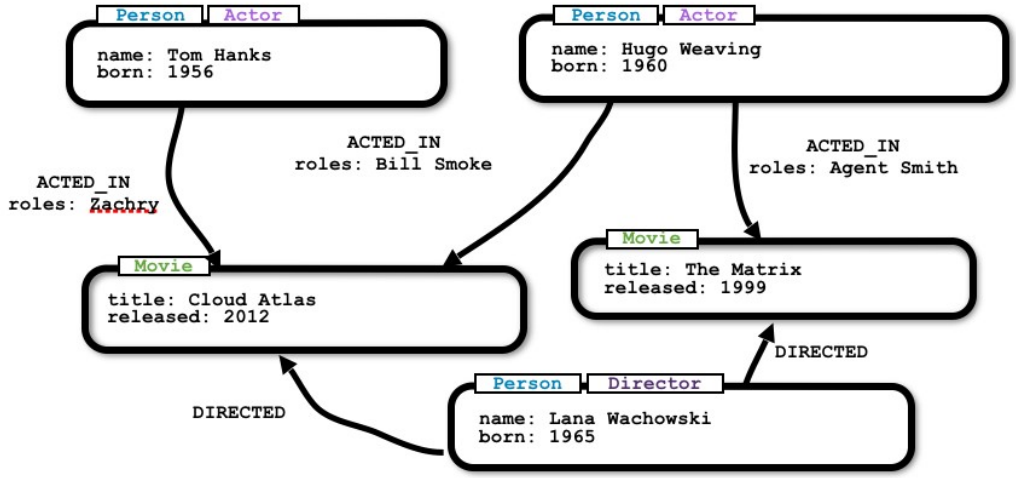
Next, we formalize our entities a bit and match expected syntax for relationship types to create the node/relationship view for the property graph model.

Matrix – match node and relationship format of property graph model



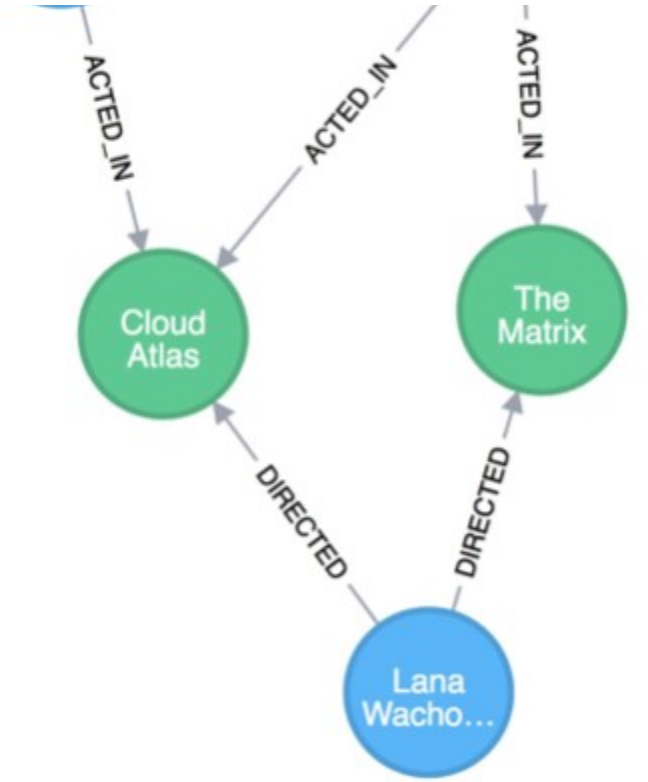
For our next step, we add labels and determine properties of our nodes and relationships for the property graph model.

Matrix – add labels and properties



Finally, we can view this data model in Neo4j and ensure it matches what we drew on the whiteboard. Also, notice how it is nearly identical to the whiteboard model we initially designed.

Matrix – final model in Neo4j



The ability to easily whiteboard your data model makes the graph data model incredibly simple and visual. There is no need to draw up business model versions or explain ERD terms to business users. Instead, the graph data model is easily understood by anyone.

Describing a Domain

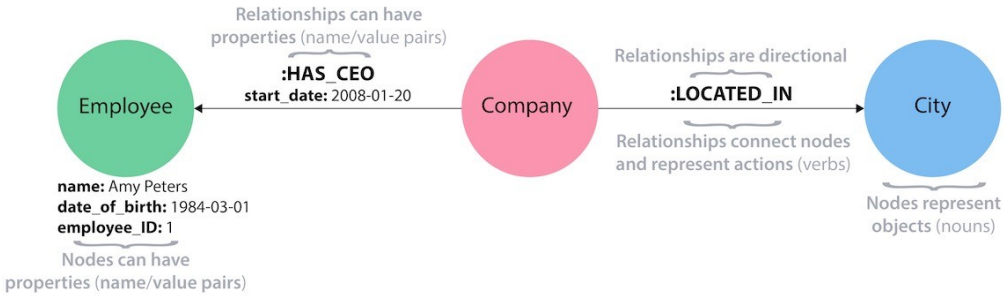
To better understand the process of designing a graph data model, let us take an example domain for a small set of data and walk through each step of how to create a graph data model from it. Consider the following scenario describing our example data entities and connections.

Scenario

*Two people, **Sally** and **John**, are friends. Both **John** and **Sally** have read the book, **Graph Databases**.*

We can use the information in this statement to build our model by identifying the components as labels, nodes, and relationships. Let us take the scenario into pieces and define them as parts of our property graph model.

Review – Property Graph Elements (click to zoom)



Nodes

The first entities that we will identify in our domain are the nodes. Nodes are one of two fundamental units that form a graph (the other fundamental unit is relationships).

Nodes are often used to represent entities, but can also represent other domain components, depending on the use case. Nodes can contain properties that hold name-value pairs of data. Nodes can be assigned roles or types using one or more labels.

> You can often find nodes for the graph model by identifying nouns in your domain. Entities such as a car, a person, a customer, a company, an asset, and others similar can be defined as nodes for a good starting point.

We can identify nodes as entities with a unique conceptual identity. In our scenario we began for Sally and John, these entities are outlined below in bold.

Scenario – Defining Nodes

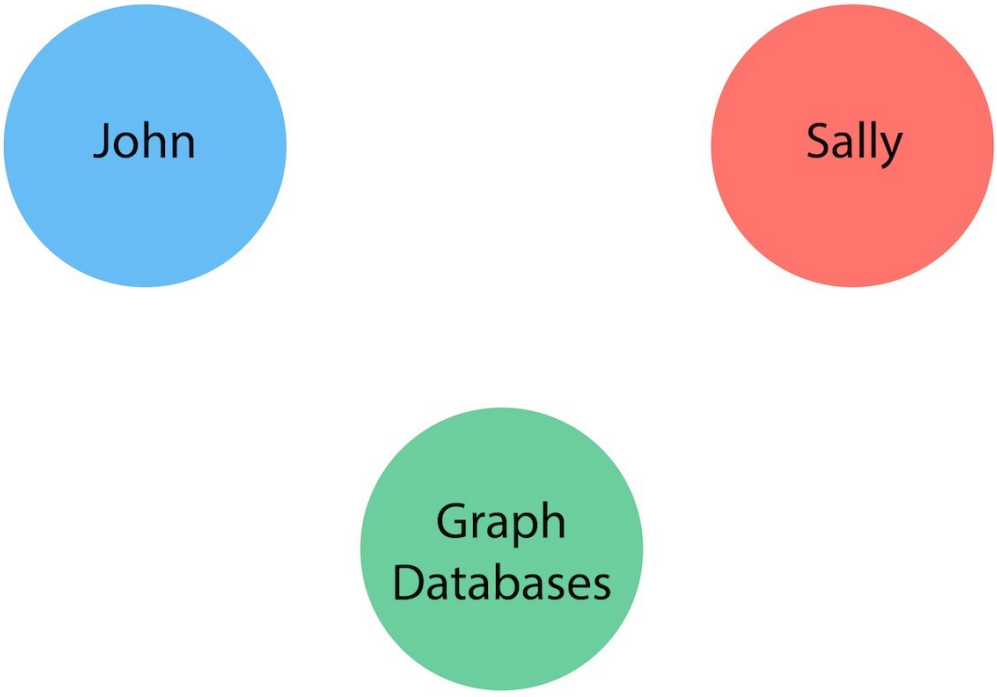
*Two people, **John** and **Sally**, are friends. Both **John** and **Sally** have read the book, **Graph Databases**.*

Extracting the nodes:

* **John**
* **Sally**
* **Graph Databases**

> Remember that a graph database takes each instance of an entity as a separate node (John and Sally would be two separate nodes, even though they are both people), and Graph Databases would be a separate node from another book.

Graph Model – Nodes



Labels

Now that we have an idea of what our nodes will be, we can decide what labels (if any) to assign our nodes to group or categorize them. The definition from Neo4j's developer manual in the paragraph below best explains what labels do and how they are used in the graph data model.

A label is a named graph construct that is used to group nodes into sets. All nodes labeled with the same label belongs to the same set. Many database queries can work with these sets instead of the whole graph, making queries easier to write and more efficient. A node may be labeled with any number of labels, including none, making labels an optional addition to the graph.

> Similar to how we found the nodes for our graph model by identifying the nouns in our scenario, you can identify labels by generic nouns or groups of persons, places, or things. General nouns that fit groups of items such as Vehicle, Person, Customer, Company, Asset, and similar terms can be used as labels in your graph.

To find out if we can group objects in our Sally and John scenario, we will start by identifying the roles of our nodes (John, Sally, Graph Databases) mentioned in the statement. We can find two different types of objects in the statement, which are emphasized below.

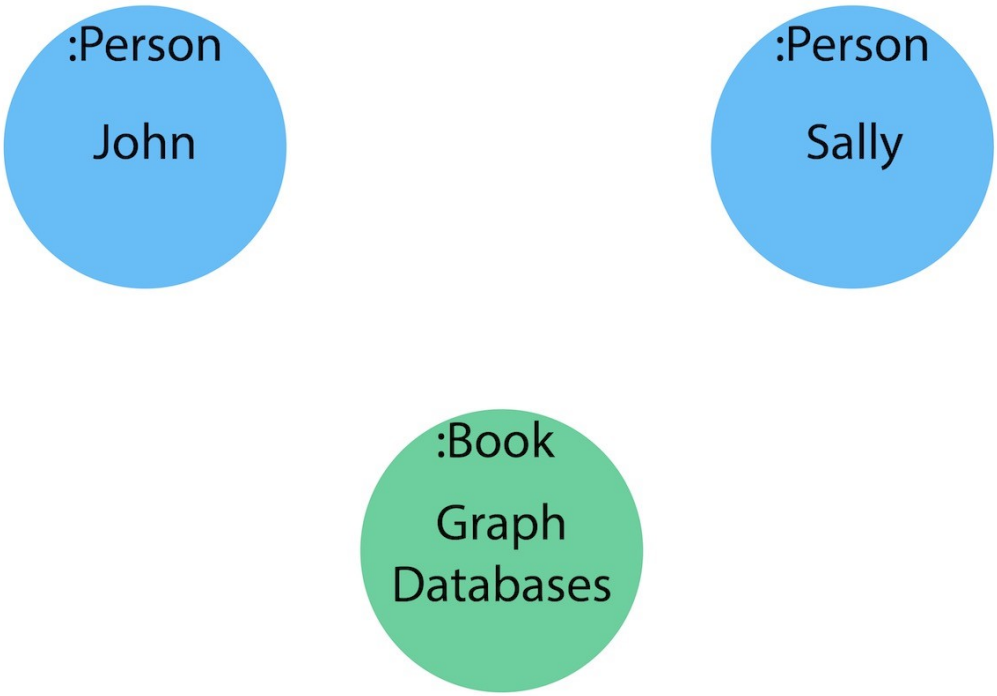
Scenario – Defining Labels

Two people, John and Sally, are friends. Both John and Sally have read the book, Graph Databases.

Extracting the labels:
* *Person*
* *Book*

Now that we have identified both our nodes and labels, we can update our graph data model to assign the labels to the nodes they describe. For **John** and **Sally**, we apply the role *Person*. For **Graph Databases**, we apply the role *Book*.

Graph Model – Labels



Relationships

We now have our main entities and a way to group them, but we are still missing one vital piece of a graph database model – the relationships between the data!

A relationship connects two nodes and allows us to find related nodes of data. It has a source node and a target node that shows the direction of the arrow. Although you must store a relationship in a particular direction, Neo4j has equal traversal performance in either

direction, so you can query the relationship without specifying direction.

The one core, consistent rule in a graph database is **"No broken links"**, ensuring that an existing relationship will never point to a non-existing endpoint. Since a relationship always has a start and end node, you cannot delete a node without also deleting its associated relationships.

> Just as we have found nodes and labels by looking for nouns, you can often find relationships for the graph model by identifying actions or verbs in your domain. Actions such as DRIVES, HAS_READ, MANAGES, ACTED_IN, and others similar can be defined as differenty types of relationships to exist between nodes.

Let us identify the interactions (which are underlined in our scenario below) between the **John**, **Sally**, and **Graph Database** nodes.

Scenario – Defining Relationships

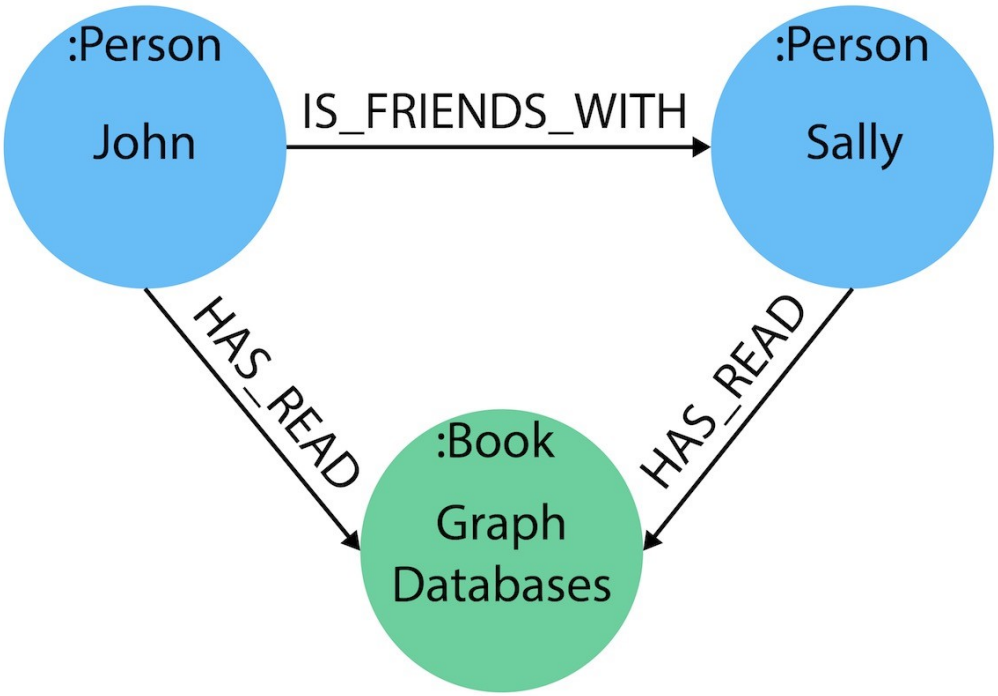
Two people, Sally and John, are friends. Both John and Sally have read the book, Graph Databases.

Relationships between nodes:
* John is friends with Sally
* Sally is friends with John
* John has read Graph Databases
* Sally has read Graph Databases

To sum up our findings, our John and Sally nodes (labeled *Person*) can be connected to each other by the is friends with relationship. John

and Sally have both read the Graph Databases book, so we can connect each of their nodes (each labeled *Person*) to the Graph Databases node (labeled *Book*) with a has read relationship.

Graph Model – Relationships



Properties

We have gone through the process of creating a basic graph data model for the interactions between people and books. We can take this data model further by defining attributes of these entities as key-value properties.

Properties are name-value pairs of data that you can store on nodes or on relationships. Most standard data types are supported as properties, with the full list published in our Developer Manual documentation.

Properties allow you to store relevant data about the node or relationship with the entity it describes. They can often be found by knowing what kinds of questions your use case needs to ask of your data.

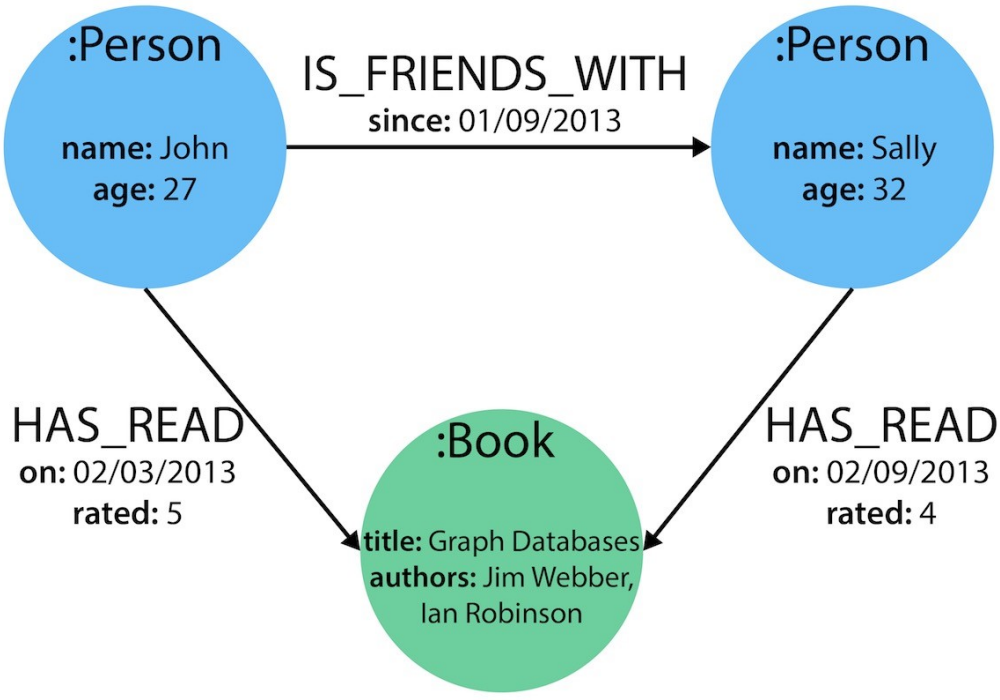
For our John and Sally scenario, we can list some questions that we might want to answer about the data.

Questions to ask of our John and Sally data model:

- When did John and Sally become friends? Or how long have they been friends?
- What is the average rating of the Graph Databases book?
- Who is the author of the Graph Databases book?
- How old is Sally?
- How old is John?
- Who is older, Sally or John?
- Who read the *Graph Databases* book first, Sally or John?

From this list of questions, you can identify the attributes that we need to store on the entities within our data model in order to answer these questions.

Graph Model – Properties



With the final model, we now can answer each of the questions we defined in our list. Of course, we can grow and change the model over

time and add/remove relationships, nodes, properties, and labels. The flexibility and simplicity of the property graph data model allows users to easily review the data structure and update it according to the changing needs of the business.

Graph Data Modeling Design

This guide is simply the introduction to data modeling using a simple, straightforward scenario. There are plenty of opportunities throughout the upcoming guides to practice modeling domains and analyzing changes to the model that might need to be made.

Every data model is unique, depending on the use case and the types of questions that users need to answer with the data. Because of this, there is no “one-size-fits-all” approach to data modeling. Using best practices and careful modeling will provide the most valuable result in producing an accurate data model that benefits your processes and use case.