Graph Databases For Enterprise Architects

Rik Van Bruggen → September 2014

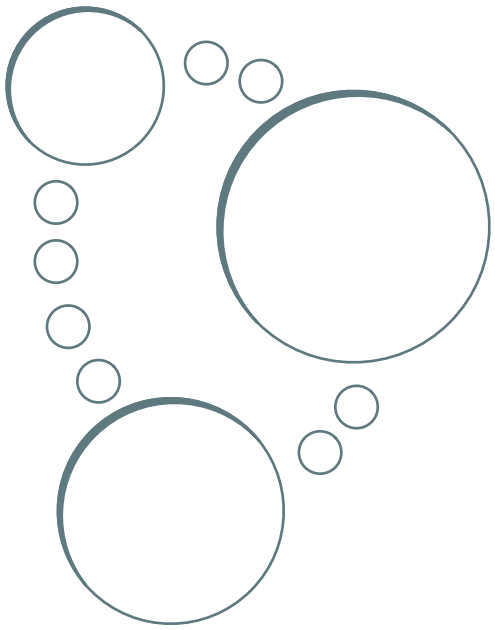


Table of Contents

Graph Databases and in the big "Data Business"

In this paper, we we would like to explore some of the concepts, trends and technologies that are surrounding us today in the age of "Big" Data. We will focus on how Graph Databases fit into these, and what the benefits will be of using these graph database technologies from the perspective of the *Enterprise Architect*. Not just developing and implementing technology for technology's sake, but for enabling a sound *Enterprise Architecture* that will stand the test of time.

Big Data is the trend; the unrelenting, ever increasing, warehouse flooding stream of data from every fleeting tweet, instant photo and look-at-me video. It's not just user content, of course. There's data about the data, real-time streams of data, and privately held business and personal data – all interwoven in an incredible web of information.

NOSQL is the answer. Rising from practical need, engineers put aside the traditional databases that were faltering, declaring that there is Not Only SQL. Amazon needed a better shopping cart, so designed Dynamo. Facebook needed to manage your life inbox, so created Cassandra. Google needed to store, well.. the Internet, so they designed Big Table. Graph databases like Neo4j arose the same way, to relate and track the flow of documents in a comprehensive content management system. From there, it evolved in a more generalistic approach to managing all kinds of semi-structured, highly interconnected data sets - to manage Graphs. As you can see, each of these solutions have broader application. In a "Big"data based business environments, data does tend to get more and more interconnected, relations between datasets become more complex - and graph based solutions become a great answer to managing this data environment's complexity.

Without wanting to discount the other NOSQL answers to the "Big" data environment (Key-Value stores, Column-Family stores, and Document Databases), we would like to explore the concept of a Graph Database now, to give us some context and understanding before we start exploring why they contribute to a better *enterprise architecture*.

Understanding Graph Databases

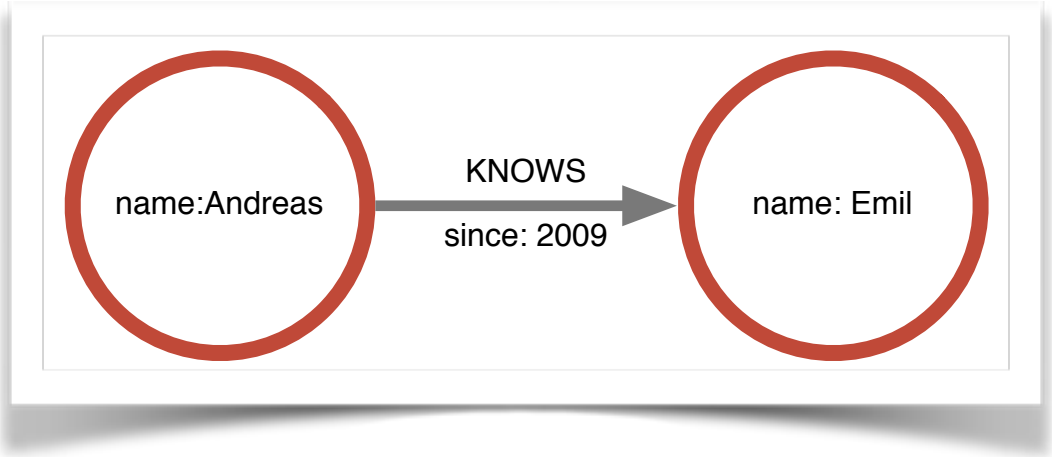
The most general-purpose of the NOSQL data models, a Graph Database can store any kind of data. Take the trees in a Document Database and connect them to each other to share common information (`Products` have `Reviews`, written by `People`), and you've got a graph. The preceding NOSQL models are special, specific cases of a graph.

Graph Database Model

The word "graph" has very context-sensitive meanings. It could refer to bar charts, or scatter plots, or even vector artwork. But when talking about the database category, think about drawing on a whiteboard with circles and lines, sketching out example data for your application; you've made a graph, so save it just as it is.

A Graph is just data that has a defined structure. Classic computer-science has many examples: lists, trees, maps, objects. All of these data structures are special cases of a Graph. A Graph is the general data structure for storing related data.

The records in a Graph database are called "Nodes" which are connected through "Relationships" that always have a direction. The traditional visual representation uses circles for Nodes and lines for Relationships. Something like this:



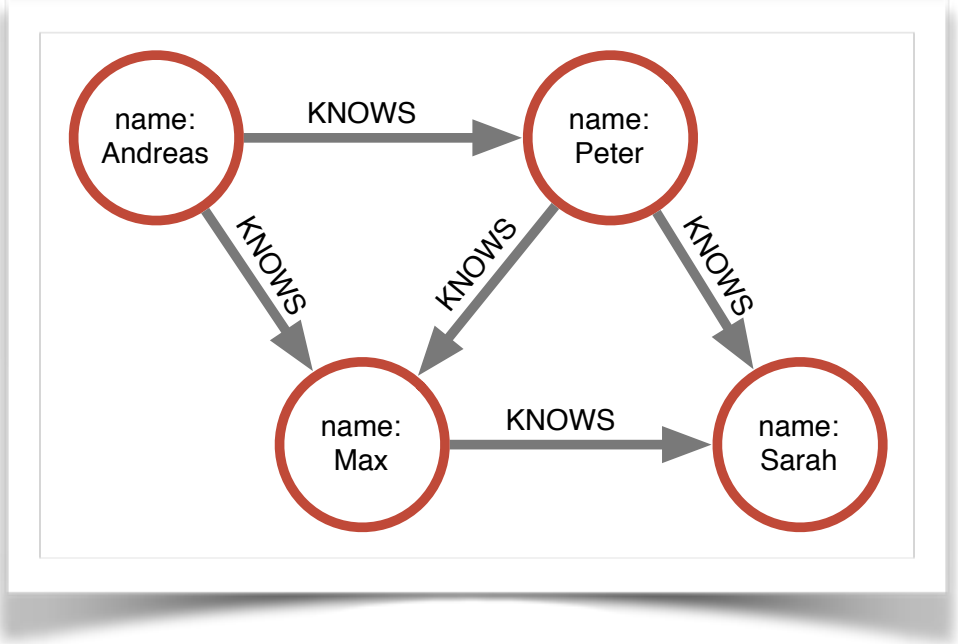
Within a database like Neo4j, both the Nodes and the Relationships can store data as "Properties" which are simple key-value pairs. In the example, both Nodes have a `name` Property, and the Relationship has a `since` Property. In addition, the Relationship is given a Type (here `KNOWS` and in all-caps, by convention) which indicates how the Nodes are related.

There is great power in this simplicity. For example, reviews can be attached to a product, and the author, who in turn can be related to other authors, so you can easily navigate back-and-forth through the data to meet new people, find similar interests, and discover related products that your friends also purchased. This navigation is how you query a Graph Database.

Querying a Graph Database

The real fun with a Graph starts when you query the database. A "Traversal" is a graph query which specifies the "Paths" to follow from a starting point. Neo4j's Cypher query language uses pattern-matching to describe a Traversal.

In this next example, we have a tiny social network. To recommend a new friend for `Andreas` we could ask the question: who are the friends of `Andreas`' friends? Looking at the Graph, the answer is intuitive: just follow along the `KNOWS` Relationships from `Andreas` to his friends, then to their friends and return those Nodes, finding `Sarah`.



The pattern match for that question looks like the shorthand you might write in an email discussing the problem. We want to find:

```
(andreas)-->(friends)-->(friends_of_friends)
```

That ascii-art is exactly what you'd use in the MATCH clause of a Cypher query:

```
START andreas=node:People(name='Andreas')
MATCH (andreas)-->(friends)-->(friends_of_friends)
RETURN friends_of_friends
```

The START clause specifies where the traversal should begin, here using an "Index" called "People" that is conveniently available. Then the RETURN specifies the result set, as you'd expect.

Similar to SQL, with some Graph-specific meaning, Cypher is comfortably familiar and sensible. You'll find the usual features: a WHERE clause for constraints, aggregation, ordering and more. For full details about Cypher, read the online Neo4j Manual.

Where are people using Graph Databases?

Of course, Neo4j is a natural choice as the default storage for any application. The Property Graph data model – Nodes and Relationships with Properties – can store and query data in the same form as it appears when you draw it on a whiteboard. No contortions or compromises required. Still, it is helpful to start with well understood use cases as you learn the best practices for working with a graph.

Social Networking

The social network, or social graph, is the most intuitive use case for a Graph Database. The expanding relationships of friends, family, neighbors, and colleagues maps neatly into a graph, where you can realize trust networks, find new friends, or understand how you're connected to someone.

Recommendations

Someone buys a book on bread baking. Could you guess what else might be a good recommendation for them to consider? Immediately, your mind may leap to baking supplies, other books by that author, or seemingly unrelated products like a garden hose that other users who bought that book typically end up buying. A graph can capture all the direct and nuanced relationships in any domain to make more meaningful recommendations.

Access Control

Consider the case of a cable-TV company: millions of customers across different regions, with different promotions, agreements, and memberships. A customer adds a sports package through the web, then flips to the game-in-progress on their TV. What hap-

pens? Hopefully, the company has that complex interleaved access control stored in a graph, so a simple query can resolve in real-time whether that channel is available.

Network Management

Imagine hosting thousands of applications, each proxied by a load balancer, spread across multiple machines, connected to routers, which are connected to other routers, all with varying power requirements. To guarantee quality service, you need a database that can make sense of the complex interconnections and dependencies, coming up with immediate answers when one piece fails so you know how to keep everything running.

Spatial

The Traveling Salesman problem, The Seven Bridges of Königsberg, and getting the best flight on Priceline are all examples of common graph problems which are framed in terms of geography and location. With Neo4j Spatial, it is possible to import Open Street Maps data and attach it to your own data, easily getting the benefit of a full GIS stack, or simple location-aware operations.

Bio-Tech

"Graphs are everywhere", has sort of become a tagline for many graph database professionals. And nowhere is this more apparent and visible as in the Bio-Tech industry. Whether you are trying to analyse protein structures, genome data, pharmaceutical dependencies, or patent implications of the latest research efforts, network oriented data-structures are almost everywhere in Bio-Tech. And Graph Databases lend themselves extremely well to modelling these networks, and extracting valuable information from them that could lead to the next live-saving drug, crop or pharma process.

Master Data Management

Organizations are rarely a neat hierarchical tree of relationships. Cross departmental relationships, overlapping sales territories, suppliers, services, contractors, and more contribute to the densely connected machinery of a modern business. A Graph Database provides the flexibility to accommodate the complexity, and a language for keeping everyone on the same page.

In General

A Graph Database is a good choice whenever you have densely connected data. These few examples are obvious, highly-connected data problems. Yet, every relational schema in the world contains at least a few join tables – each one an indication that the data isn't just related, but has a relationship. There is great power and flexibility in realising the value in relationships.

But why should the Enterprise Architect care?

All of the above are very interesting, inspiring use cases for graph databases, but you are probably thinking by know: how does that apply to me as an enterprise architect, and why is it relevant? Let's explore that.

Time to market

A modern Enterprise Architecture, these days, should enable quick business decisions. If business managers want to react to competitive threats, or better still, want to seize a new business opportunity, the architect's response cannot be that "he can deliver that in 12 months from now". That may have been ok in the past, but it's not anymore. Agility is key, and being to implement modern-day functionality like social network integration, real-time recommendations, real-time fraud detection - just to name a few examples - needs to be quick.

One of the key things that a graph database brings to the table to enable short time to market is the excellent fit with these domains. There is a natural fit between these very interconnected domains and a graph representation. There is no need to translate back and forth from the domain into the relational storage paradigm, and that alone cuts down implementation efforts by an order of magnitude. Complex operations like pathfinding, recursive questions, complex pattern matching or deep joins - all of the above are much simpler in a graph database than in any other model - thereby enabling shorter time to market for the enterprise architect.

Flexibility

We all know the cliché: "the only thing constant is change". And we have all lived its reality: in the fast moving business world that we enable with our enterprise architecture, we have to be able to act quickly, react to changing circumstances, and be flexible in the way that we react so that we can learn, adapt and improve our performance. In the enterprise architect's world, we have adopted things like "Agile" development methodologies for a reason: we understand that in the fast paced business environment, our IT systems have to be equally fast paced and able to adapt. IT systems can no longer feel like a straitjacket - they have to allow our business processes to evolve and adapt, in a flexible way.

Graph databases allow for that flexibility. Developers don't have to spend valuable time trying to model what they don't know, have the ability to grow their data models as their understanding of the problem domain grows, and can flexibly work with their data structures in many different situations. This in itself, is a huge added value to any enterprise architect who is tracking the constant change around them.

Operational performance

In the same way that graph database often "enable" certain capabilities for specific use cases as described above, they just as often offer a very significant performance improvement over existing technologies. Some query patterns - like the deep/recursive join or the pathfinding operation - require an enormous amount of hardware/software horsepower in the traditional relational database world in order to deliver the results in a timeframe that users would accept. And even then, we all know cases where the query performance would become brittle and unpredictable under load.

This is where graph databases can really help. The same queries that were causing constant headache in the relational world, would predictably run like a breeze in a graph world. That's a fantastic trait for an enterprise architecture that business managers can rely on. The fact that it will probably cost them less in terms of hardware and software -

is of course a nice bonus that we should not underestimate. Everyone understands the benefits of a sound architecture - but when it actually saves money, business managers will really start paying attention.

Conclusion

A Graph Database like Neo4j, is a NOSQL solution for making sense of the incredible connections found in all data big and small. Graphs are everywhere. When stored in a mature, proven database like Neo4j, you can comfortably navigate the complexity, and reap real, tangible rewards for your Enterprise Architecture.

Should you ever want to discuss some the topics in this whitepaper, please do not hesitate to visit www.neotechnology.com or www.neo4j.org, or email us at info@neotechnology.com .