# Graph Database: Neo4J da teoria à pratica.

**Iuri Andreazza[1], Eduardo Pereira da Silva[2], Talita Audibert[1], Paulo Gräbin[3]**

[1]Unidade academica de graduação – Universidade do Vale dos Sinos (UNISINOS)
São Leopoldo – RS – Brasil

{iuri.andreazza,eduardobursa,tali.audibert,plgrabin}@gmail.com

***Abstract.*** *This meta-paper describes the style to be used in articles and short papers for SBC conferences. For papers in English, you should add just an abstract while for the papers in Portuguese, we also ask for an abstract in Portuguese ("resumo"). In both cases, abstracts should not have more than 10 lines and must be in the first page of the paper.*

***Resumo.*** *Este artigo gerencia*

## 1. Introdução

What is a Graph Database?

A Graph Database is not a database to store graphics or images as its name may suggest. It?s a database that uses graph structures such as nodes, properties and edges to represent and store data. In addition, it allows you to represent any kind of data without the limitation of regular databases.

To better understand, let?s talk about each of these structures individually and what they represent:

Node: A node can be used to represent any type of entity that you can think of, be it a business, a blog post, a location, an oil rig, a city, etc. Graph databases don?t care what type of data they?re representing.

Properties: Properties, sometimes called attributes, are named values that relate to nodes. For example, if we take into consideration our City representation of a node, one of the properties would be ?name?, another would be ?population?, and so on.

Edges: Edges, sometimes called Relationships, connect nodes-to-nodes and organize them into arbitrary structures such as a Map, List or a Tree. It?s important to note that when a node is the start node of a relationship, the relationship from that node?s perspective will be an outgoing relationship. And when a node is at the end of a relationship, the relationship from that node?s perspective will be an incoming relationship. Understanding this will make it easier for you to follow the examples.

What can a Graph Database be used for?

There are many scenarios for which one could consider using a Graph Database. Before I list a few examples, as with any architectural and technical decision, you need to analyze all possible solutions. Then, you can select those that are best for you, according to your specifications.

Some of these scenarios include: social networking, fraud detection, people/movie/music recommendation, manufacturing, etc. Using the social networking example, it would be somewhat trivial to do something like ?given the fact that Bob is my

friend, give me all friends that are friend?s of friend?s of friend?s of Bob?. This is possible because of the path-finding algorithms involved are easy to implement by traversing through the graph. Imagine doing that through a relational database? A nightmare!

Another advantage when using a Graph database is that you can easily and more naturally model a domain using a whiteboard or a piece of paper. Specifically, nouns that are used become nodes, verbs become relationships, and adjectives and adverbs become properties.

Graph Database Example

Let?s examine the following scenario and how we would structure it using a Graph Database: John is friends with Bob, and Bob is friends with Mark. Visually, this is how we could represent the scenario:

You notice that the visual representation is pretty much how we verbally expressed our scenario. This is a very simple example that we can use to compare the implementation of a traditional SQL database structure versus a Graph database structure:

Traditional SQL

In a traditional SQL structure, each row in the users table represents a user, and each row in the friends table represents a relationship between two users. If we decide to add additional properties to the users table at a later time, we would have to alter the base structure of that table. And if we wanted to add new properties to only a subset of users we would still have to alter the entire users table, or create a new table to accommodate the new values just for the subset of users. Not an ideal scenario when dealing with tens of millions of records.

Graph Database

Graph Database Structure

On the other hand, a Graph database has no set structure or schema for the data, much like a NoSQL database. Now, let?s understand what the graph above is representing: each node represents an entity ? a User in our case; each node contains property values, in this case it?s the User?s name; and each line represents a relationship between the nodes. This is as simple as it gets. And to complement the scenario described in the traditional SQL example, if we did decide to add additional properties to a subset of users, we could easily perform this action on a per-node level instead of a table-wide transaction.

What is Neo4j

Neo4j is ?The World?s Leading Graph Database? according to Neo Technology, the developer behind the project. It?s a commercially supported open-source graph database implemented in Java and some of the key characteristics include:

Data representation using an intuitive graph-oriented model. Binding for a number of languages, including Ruby, Python and Closure. Disk-based, native storage manager completely optimized for storing graph structures for maximum performance and scalability. Massive scalability, it can handle graphs of several billion nodes/relationships/properties on a single machine. Easy to use and convenient object-oriented API. Handles large graphs that don?t fit in memory with durability and a fully

persistent transactional store. Powerful traversal framework for high-speed traversals in the node space. Optimized for highly connected data. Small footprint, only about 750k jar file. It provides a REST interface for languages not supported by its bindings. It?s capable of traversing depths of over 1000 levels at millisecond speeds. It provides a dual license: open source and commercial. It integrates seamlessly with Lucene, providing full-text search to nodes and relationships, including phrase queries, wildcard queries, proximity queries, ranked searching, sorting, and more.

### 1.1. Organização

Este artigo está organizado da seguinte maneira, tendo que, na sessão 1 temos o trecho introdutório,

## 2. Conceito

## 3. Abordagem Prática

## 4. Trabalhos relacionados

## 5. Conclusão

## 6. References

Bibliographic references must be unambiguous and uniform. We recommend giving the author names references in brackets, e.g. [Knuth 1984], [Boulic and Renault 1991], and [Smith and Jones 1999].

The references must be listed using 12 point font size, with 6 points of space before each reference. The first line of each reference should not be indented, while the subsequent should be indented by 0.5 cm.

## References

Boulic, R. and Renault, O. (1991). 3d hierarchies for animation. In Magnenat-Thalmann, N. and Thalmann, D., editors, *New Trends in Animation and Visualization*. John Wiley & Sons ltd.

Knuth, D. E. (1984). *The TEX Book*. Addison-Wesley, 15th edition.

Smith, A. and Jones, B. (1999). On the complexity of computing. In Smith-Jones, A. B., editor, *Advances in Computer Science*, pages 555–566. Publishing Press.