Graph Database: Neo4J teoria e uso.

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

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

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

Abstract. *This article aims to provides an overview about the core concepts of a no-SQL database that uses graphs as representative model and operations using another language rather than SQL to fetch data. It's also reviewed the pratics from the creation of the project to some basic queries with some data to illustrate your queries and functionality. Finally it is possible to view the use of this database and how we can have big gains when use it correctly.*

Resumo. *Este artigo traz uma visão geral dos conceitos de um banco no-SQL que utiliza grafos como modelo representativo dos seus dados e operações e utilizando de uma outra linguagem do que o SQL para consultas aos seus dados. Também é tratado a utilização, desde a criação do projeto até algumas consultas basicas em alguns dados para exemplificar suas consultas. Por fim é possivel visualizar o uso deste tipo de banco de dados e como podemos ter grandes ganhos quando utilizarmos corretamente.*

1. Introdução

1.1. Organização

Este artigo está organizado da seguinte maneira, tendo que, na sessão 1 temos um geral do artigo e a introdução do assunto, já na sessão 2 é tradado dos conceitos básicos do banco de dados como também um geral da teoria dos grafos que está sendo usado. Na sessão 3 é apresentado o uso prático do banco de dados desde a criação do projeto até sua utilização dentro da linguagem *java*, durante a sessão **??** é apresentado os trabalhos relacionados com este artigo e ao final na sessão 4 é mostrado quais beneficios que podemos ter ao utilizar esse tipo de abordagem.

2. Conceito

Um banco de dados orientado a grafo não é um banco de dados para armazenar gráficos ou imagens como o nome pode sugerir. É um banco de dados que usa estruturas de grafos, como nós, propriedades e arestas para representar e armazenar dados. Além disso, ele permite que você represente qualquer tipo de dados, sem a limitação de bases de dados regulares, ou seja os relacionais tradicionais.

Para entender melhor, vamos falar sobre cada uma dessas estruturas individualmente eo que eles representam, possivel visualizar em um exemplo simples na figura 1:

Nó: Um nó pode ser usado para representar qualquer tipo de entidade, seja ele uma empresa, um blog, um local, uma plataforma de petróleo, uma cidade. Um bando de dados orientado a grafos não se importa com o tipo de dados que se está sendo operado.

Propriedades: Propriedades, às vezes chamados de atributos, são os valores nomeados que dizem respeito a nós. Por exemplo, se levarmos em consideração a representação da cidade em um nó, uma das propriedades seria ”nome”, outra seria ”população”, e assim por diante.

Arestas: Arestas, às vezes chamados de Relacionamentos, serve para conectar os nós e organizá-los em estruturas arbitrárias, como um mapa, lista ou uma árvore. É importante notar que quando um nó é o nó de início de uma relação, a relação do ponto de vista de que o nó será uma relação de saída, porém quando um nó está no fim de uma relação, será uma relação de entrada.

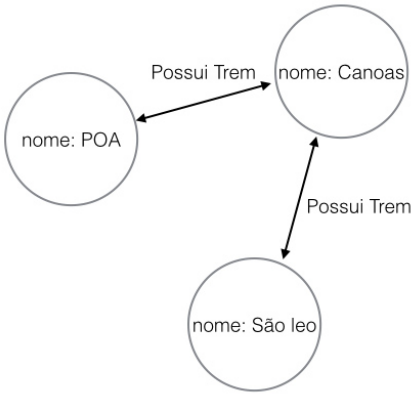

Figure 1. Estrutura base do grafo representando cidades ligadas por uma linha de trem

2.1. SQL Tradicional

Em uma estrutura SQ: tradicional, cada tupla na tabela representa uma informação na qual pode ser relacionada com outras tuplas em outras tabelas. Se for decidido adicionar propriedades para as tabelas em outro momento será necessário usar instruções DML para alterar a estrutura da tabela e tratar caso a caso em caso de existencia de restrições e outros eventos relacionados com suas propriedades e mesmo que se fosse preciso adicionar propriedades apenas para algumas tuplas da tabela será necessário igualmente alterar a tabela ou mesmo criar uma nova estrutura para comportar esse novo conjunto de informações, este cenário não é bom quando é existem milhões de registros que necessitam de dados especificos.

2.2. Graph Database

Neo4j é a applicação lider em banco de dados orientados à grafos sendo suportado comercialmente e tendo sua versão *open-source* como fonte de fomento para novos usos assim chamando a comunidade para dar suporte a ferramenta. As principais caracteristicas tem como base a representação intuitiva de dados usando um modelo orientado a grafo. A portabilidade para um número de linguagens de programação, incluindo Ruby, Python e Java. Capacidades de um SGBD, armazenamento nativo completamente otimizado para armazenar estruturas dos grafos para com o máximo de desempenho e escalabilidade. Escalabilidade maciça, podendo lidar com grafos de vários bilhões de nós / relações /

propriedades em uma única máquina. Fácil de usar e conveniente possui uma API orientada a objetos. Com grafos grandes não cabem em memória, utiliza um mecanismo de persistencia de alta velocidade que possibilita varreduras em nós em perda de desempenho com o mecanismo transacional funcional. Quadro travessia poderosa para travessias de alta velocidade no espaço de nó. Otimizado para dados altamente conectados. Pegada pequena, somente cerca de 750k arquivo jar. Ele fornece uma interface REST para linguagens não suportadas nativamente. É capaz de atravessar mais de 1000 níveis de profundidade em velocidades de milissegundos e se integra perfeitamente com o Lucene, proporcionando pesquisa de texto completo para nós e relacionamentos, incluindo consultas de frase, consultas curinga, consultas de proximidade, a pesquisa classificatórias e muito mais [Team 2014b].

Existem vários cenários onde o banco de dados orientado a grafos pode ser usado. Antes de listar alguns exemplos, com qualquer decisão arquitetônica e técnica, deve-se analisar todas as soluções possíveis, caso nenhuma representa um ganho significativo ou uma simplicidade de uso, ou a mesma exigir a flexibilidade de representação do dado, pode-se pensar em adotar um banco orientado a grafos.

Alguns desses cenários incluem: redes sociais, detecção de fraudes, recomendações de pessoas/filmes/músicas, manufatura, etc. Usando uma rede social como exemplo, seria de alguma forma trivial fazer algo como ?dado o fato de que Bob é meu amigo, dê-me todos os amigos que são amigos dos amigos dos amigos de Bob?. Isso é possível pois os algoritmos de caminho de busca envolvidos são fáceis de implementar atravessando os grafos.

Outra vantagem usando banco de dados orientado a grafos é que você pode fácil e naturalmente modelar um domínio usando uma lousa ou um pedaço de papel. Substantivos usados tornam-se nodos, verbos tornam-se relações e adjetivos e advérbios tornam-se propriedades.

2.2.1. Estrutura da base de dados

As unidades fundamentais que formam um grafo são nodos e relacionamentos. Em Neo4j tanto nodos quanto relações podem conter propriedades. Os relacionamentos entre nodos é uma parte chave de um banco de dados orientado a grafos. Eles permitem buscas relacionadas aos dados. Um relacionamento conecta dois nodos, e é a garantia de ter um nodo de inicio e fim válidos. Um relacionamento sempre tem uma direção, pode ser visto como saída e entrada relacionado à um nodo, que é útil quando percorremos um grafo. Embora as relações sempre tem uma direção, você pode ignorar o sentido em que não é útil em sua aplicação. Note que um nodo pode ter um relacionamento com ele mesmo. Para melhorar ainda mais o gráfico transversal, todas as relações têm um tipo de relacionamento. Note que a palavra tipo é uma classificação.

Propriedades são pares de chave-valor onde o chave é uma string. Os valores das propriedades podem ser tanto um valor primitivo quanto um array de um tipo primitivo. Por exemplo String, int e int[] [Team 2014a].

Uma classificação é um um grafo nomeado usado para agrupar nodos em conjuntos; todos os nodos classificados são com a mesma classificação pertence ao mesmo

conjunto. Muitas consultas podem funcionar com esses conjuntos em vez de todo o grafo, fazendo consultas mais fáceis de escrever e mais eficientes. Um nodo pode ser classificado com varios numeros de classificações, mesmo nenhum. Fazer classificações é opcional no grafo.

Um caminho é quando um ou mais nodos que se conectam através de relacionamentos, geralmente recuperados como uma consulta ou um resultado transversal.

Atravessando um grafo significa visitar seus nodos, seguindo as relações de acordo com algumas regras.

Neo4j é um banco de dados orientado a grafos com esquemas opcional. Você pode usar o Neo4j sem qualquer esquema. Você pode introduzi-lo opcionalmente para ganhar performance ou beneficios de modelagem [Bachman 2014].

2.2.2. Resumo das relações

Abaixo é listado as relações que dentro de um banco de dados orientado a grafos pode ter:

- Um Grafo grava dados em Nodos os quais tem Propriedades.
- Nodos são organizados por Relacionamentos os quais também tem Propriedades.
- Nodos são agrupados por Classificações dentro de Conjuntos.
- Uma Transversal navega em um Grafo; Identifica caminhos os quais ordenam Nodos.
- Um índice mapeia de propriedades tanto para Nodos ou Relações
- Um Banco de Dados Orientado a Grafos gerencia um Grafo e também gerencia Índices relacionados.

3. Abordagem Prática

Vamos examinar o seguinte cenário e como estruturariamos usando um banco de dados orientado a grafos: Jonny é amigo de Paulo, e Paulo é amigo de Duda. Visualmente é assim que poderiamos representar o cenário:

Note que a representação visual é basicamente como nos expressamos verbalmente, visivelmente demonstrado na figura 2, sendo um bom exemplo simples que podemos usar para comparar a implementação de um banco de dados tradicional versus um banco de dados orientado a grafos.

Em uma estrutura SQL tradicional, cada linha na tabela de usuários representa um usuário, e cada linha na tabela de amigos representa uma relação entre dois usuários. Se decidirmos adicionar propriedades na tabela de usuários mais tarde, teriamos que alterar a estrutura da tabela. E se quiséssemos adicionar uma propriedade nova em um subconjunto de usuários apenas, mesmo assim teriamos que alterar toda a estrutura da tabela usuários, ou criar uma tabela nova para acomodar os valores novos somente para o subconjunto de usuários. Não é um cenário ideal quando lidamos com dezenas de milhões de registros [Bachman 2014].

Por outro lado, um banco de dados orientado a grafos não tem um conjunto de estrutura ou esquema para os dados, basicamente é um banco NoSQL. Vamos entender o que o gráfico acima representa: cada nodo representa uma entidade (nesse caso um

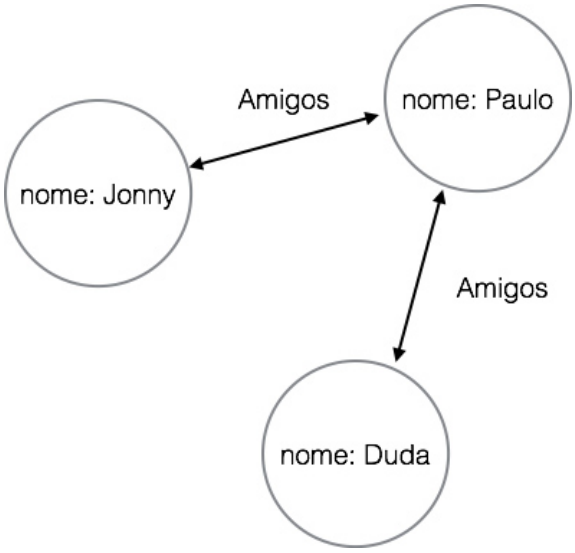

Figure 2. Estrutura do grafo representando ligações de amizade entre amigos

usuário) 3. Cada nodo contém valores das propriedades (nesse caso é o nome do usuário) e cada linha representa a relação entre os nodos. E para complementar o cenário descrito no exemplo do SQL tradicional, se decidirmos adicionar propriedades em um subconjunto de usuários, poderiamos facilmente executar essa ação em nível de nodo em vez de uma transação de uma tabela grande [Williams 2014a].

A codificação para uso disto é demasiado simples, como demostrado abaixo é possivel criar o grafo utilizando a estrutura adequada e realizar buscas no mesmo utilizando os atributos e relações que foram dados. O trecho de codigo exibido abaixo demonstra o uso para busca de pessoas que são amigos de uma pessoa que tenha no nome *Jonny*, nesta facilidade de tratamento de dados as possibilidades existentes são muitas, sendo que é possivel criar um modelo de dominio orientado a relacionamentos [Williams 2014b].

```
for (Relationship r : pessoas.getRelationships(amigos)) {
  if ((int) r.getProperty("nome").contains("Jonny")) {
    Node primoAmigo = r.getStartNode(); //Algo com o node
  }
}
```

4. Conclusão

Ao utilizar um banco de dados orientado a grafos é possivel notar que se a necessidade de representação é em demasia abstrata ou não administravel em seus requisitos, podemos adotar este modelo na qual é possivel extrair relações e funcionamentos no qual um banco de dados relacional tradicional não seria possivel ou no qual se perderia muito tempo criando a representação funcional destes dados. Sua utilziação é facil e rápida tendo suporte a várias plataformas até mesmo para linguagens que não são consideradas nativas

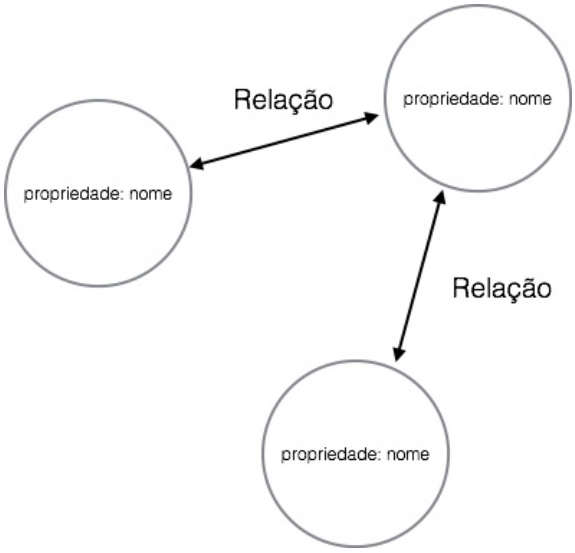

Figure 3. Estrutura do grafo representando ligações de amizade entre amigos

é possivel extrair beneficios do mesmo. Contudo não é indicado e nem adequando o uso do mesmo na expressão de relações que são superficiais ou simplistas, isto pode causa um *overhead* no uso e uma performance baixa.

References

Bachman, M. (2014). Modeling data in neo4j: Qualifying relationships. Disponível em http://java.dzone.com/articles/modelling-data-neo4j-0.

Team, N. (2014a). Java tutorials. Disponível em http://docs.neo4j.org/chunked/stable/tutorials-java-embedded-hello-world.html.

Team, N. (2014b). Projeto graph database. Disponível em http://neo4j.com/docs/2.0.3/.

Williams, A. (2014a). Neo4j, a graph database for building recommendation engines, gets a visual overhaul. Disponível em http://techcrunch.com/2014/02/02/neo4j-a-graph-database-for-building-recommendation-engines-gets-a-visual-overhaul/.

Williams, A. (2014b). Neo4j, a graph database for building recommendation engines, gets a visual overhaul. Disponível em http://techcrunch.com/2014/02/02/neo4j-a-graph-database-for-building-recommendation-engines-gets-a-visual-overhaul/.