

Aula 9: Modelo de dados NoSQL baseado em grafos – parte I

Apresentação

Estudaremos o modelo de banco de dados em grafos, muito útil em aplicações que manipulam muitos relacionamentos em seus dados, tal como é caso das redes sociais, por exemplo.

Usaremos no estudo o banco de dados Neo4J, que possui uma interessante interface gráfica e tem código aberto.

Objetivo

- Identificar as características do modelo de banco de dados baseado em grafos;
- Listar as vantagens e tipos de aplicações indicadas para uso do banco de dados em grafos;
- Escrever comandos NOSQL para a definição da estrutura do banco de dados em grafos.

Introdução

Muitos tipos de aplicações recentes apresentam uma modelagem de dados complexa e que torna o projeto no modelo relacional muito mais difícil de ser implementado.

Pelas características dos bancos de dados relacionais, seriam necessários muitos relacionamentos entre tabelas para responder às consultas de uma aplicação como, por exemplo, uma rede social ou os múltiplos relacionamentos existentes entre os consumidores e os tipos de produtos de seus interesses.

Dos modelos de banco de dados NOSQL vistos na disciplina até o momento:

Os modelos chave-valor e orientado a documentos são destinados para aplicações que necessitem de uma forma facilitada de armazenamento e de eficiência para a recuperação dos dados, tal como um arquivo de documentos.

Já o modelo de dados com o armazenamento orientado em colunas privilegia a performance em relação ao uso de projetos de aplicações mais simples no tocante à sua modelagem.

Assim, houve a necessidade da criação de um modelo de dados mais flexível (entre outras importantes características) para atender às aplicações mais recentes. Nesse contexto, surgiu o modelo de dados baseado em grafos.

Os bancos de dados baseados em grafos surgiram ainda nos anos 1980 e 1990, juntamente com o modelo orientado a objetos, porém, logo perdeu espaço para os bancos de dados semiestruturados com a linguagem XML (eXtensible Markup Language).

Com o avanço da chamada Web de Dados (ELMASRI; NAVATHE, 2018) e a popularização das redes sociais, ressurgiu o interesse nos sistemas gerenciadores de banco de dados (SGBDs) não relacionais. Para o estudo desse modelo, iremos usar o SGBD [Neo4J](#). Lançado em 2007, hoje é o mais usado dos SGBDs que seguem o modelo de dados em grafos.

Comentário

Mapearemos as características do banco de dados baseado em grafos (SADALAGE; FOWLER, 2013), listaremos as vantagens e tipos de aplicações indicadas para esse tipo de modelo e, por fim, escreveremos comandos NOSQL para a definição da estrutura do banco de dados em grafos.

Banco de dados baseado em grafos

Características

O banco de dados é composto por dois tipos de componentes principais, como em toda estrutura em grafos: Os vértices e as arestas.

Vértices

Os vértices são também chamados de nós. Os nós são usados para a representação de entidades, de forma semelhante ao conceito de entidades no modelo relacional, o que seria a representação das tabelas. Assim, como exemplos de entidades, podemos citar as pessoas e lugares.

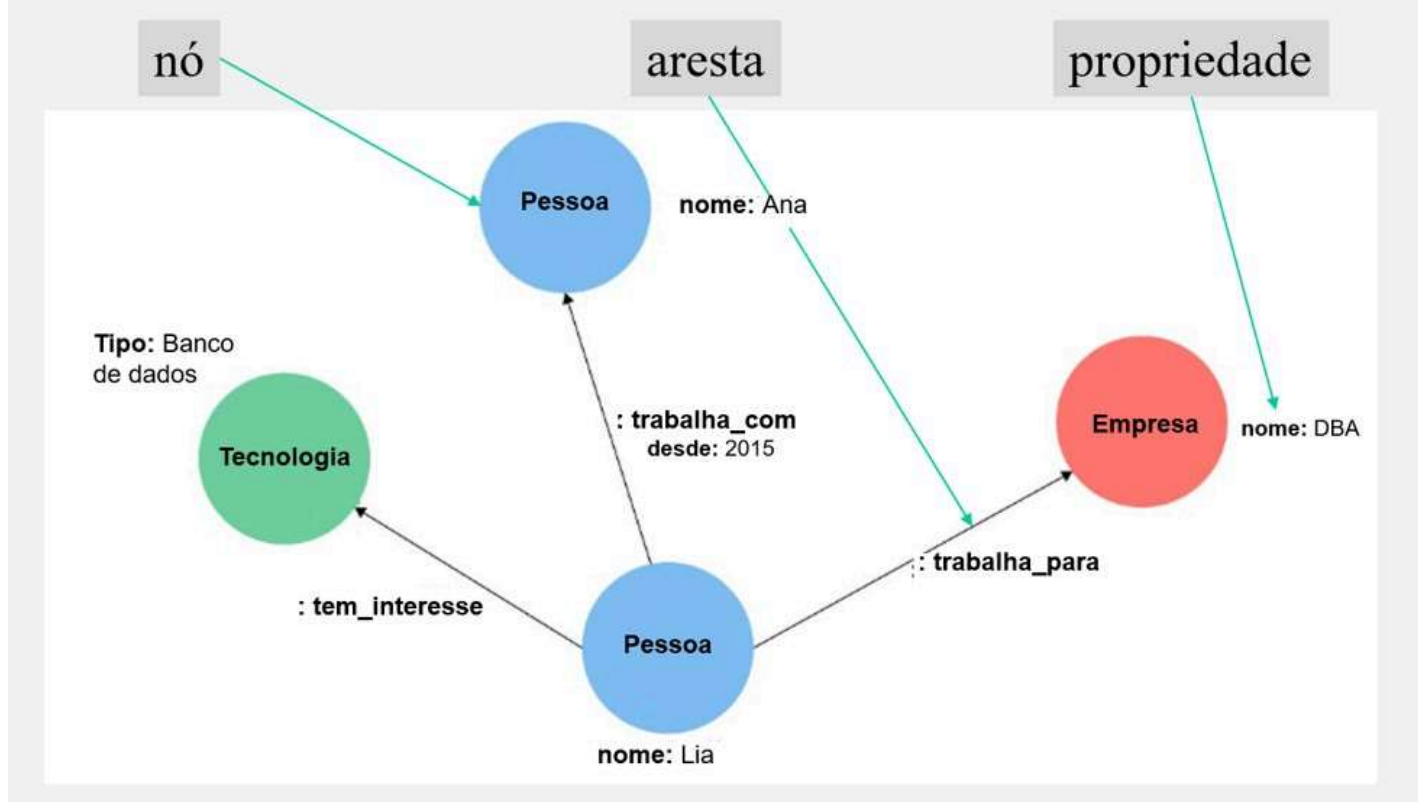
Os nós são as ocorrências das entidades no gráfico, e podem conter qualquer número de atributos (formados por pares de chave-valor), chamados de propriedades. Esses nós podem ser sinalizados com marcadores ou cores que representam suas diferentes funções no domínio do projeto. Os nós e relações suportam propriedades, um par de valores-chave onde os dados serão armazenados.

Arestas

As arestas são conexões entre os nós e representam os relacionamentos entre os vértices ligados por essas arestas. A Figura 1 indica um exemplo para o relacionamento de trabalho e de interesses entre duas pessoas em uma companhia. Mesmo com esse exemplo simples, podemos identificar o nó, a aresta e as propriedades dos nós.

Para facilitar a identificação das entidades, cores são usadas para cada uma delas.

Figura 1 - Representação de um modelo em grafos.



 Adaptado de: neo4j.com.

Relacionamento entre nós

Como no modelo de entidades e relacionamentos, estes são nomeados de modo a traduzir a semântica da ligação entre duas entidades, como, por exemplo, o empregado *trabalha_para* a empresa. Os relacionamentos se apresentam sempre únicos em direção, tipo, com um nó inicial e um final.

Assim como no caso dos nós, os relacionamentos têm também uma lista de propriedades. Essas propriedades podem indicar tipos como um valor temporal (como no exemplo, desde: 2015 ou pesos, valores, distâncias, classificações ou valores de intensidade).

Entre as principais características do modelo, pode-se destacar a performance do banco de dados, tendo em vista o modo de acesso nativo aos nós e relacionamentos, permitindo percorrer rapidamente milhares de nós por segundo.


Para isso, o SGBD faz uso do poder da computação distribuída. Dois nós podem compartilhar qualquer número de relacionamentos sem prejudicar o desempenho do banco de dados.

Vantagens e tipos de aplicações

Até aqui, foi visto que a estrutura fundamental do modelo de banco de dados em grafos é chamada de "relacionamento de nós". Essa estrutura é muito adequada quando a aplicação necessita lidar com dados altamente conectados. Apesar de serem armazenados em uma direção pré-determinada, os relacionamentos podem ser sempre navegados de um modo eficiente em qualquer direção.

Esse modelo de dados facilita a navegação pelos relacionamentos. Vale também destacar que esse tipo de armazenamento e navegação não é eficiente em SGBDs relacionais devido às estruturas de tabelas rígidas e a dificuldade de seguir as conexões entre os dados, a não ser por operações de junção, custosas em tempo de execução.

Exemplos de aplicações

 Clique no botão acima.

Como exemplos de aplicações indicadas para serem desenvolvidas com o modelo de grafos, podem ser listadas as aplicações para o gerenciamento de **dados geográficos** ou para modelar **redes de computadores** de um provedor de telecomunicações.

Um outro exemplo que pode ser estendido para representar a riqueza de possibilidades de relacionamento é a aplicação do modelo nas redes sociais. Nessa aplicação, os vértices serão os usuários e as arestas podem indicar o relacionamento dos seguidores da rede.

Normalmente, o banco de dados baseado em grafos será o mais indicado para **aplicações científicas e técnicas**.

Um exemplo de uso é o voltado para o tratamento de aplicações complexas, como as que visam o mapeamento para **perfuração de campos de petróleo** em tempo real, as quais podem se beneficiar do uso do modelo de grafos.

Nesse tipo de aplicações, muitas variáveis são usadas, como, por exemplo, o valor da temperatura, a salinidade do mar da região, assim como as propriedades biológicas, químicas e físicas.

Os modelos de doenças e interações genéticas, bem como pesquisas de padrões gráficos de proteínas para encontrar outros genes que podem estar associados a uma determinada doença, são exemplos de aplicações científicas que podem ser desenvolvidas em grafos.

Fazendo uso de um modelo baseado em uma plataforma apoiada na computação distribuída, ao invés de um ambiente relacional, aplicações complexas podem ser executadas em menos tempo.

Para encerrar, um outro exemplo é o de uma aplicação para trabalhar com o cruzamento de dados distintos, mas relacionados, permitindo analisar melhor as possibilidades de uso.

Por exemplo, a verificação de tipos de relacionamentos entre assuntos atuais nas redes sociais e a procura por produtos relacionados e o consequente aumento das vendas desses determinados produtos; ou, ainda, as preferências de produtos baseadas em círculos de amizades, são aplicações típicas dos chamados sistemas de recomendação.

Para o estudo do banco de dados baseados em grafos, usaremos o SGBD [Neo4J](#). O software é um projeto de código aberto, licenciado sob a licença pública GNU. O Neo4J tem suporte de transações ACID (Atomicidade, Consistência, Isolamento e Durabilidade) e tem também como característica a alta disponibilidade por conta do funcionamento em computação distribuída.

O banco de dados possui um processo de escalonamento e apresenta facilidade para ser modelado por conta da estrutura fundamentada nas propriedades dos relacionamentos dos nós nele contidos. O banco de dados no Neo4J não requer o uso de um esquema.

Comentário

Entre as desvantagens do banco de dados em grafos podemos citar, por exemplo, o fato de que os nós não têm a possibilidade de criação de um auto relacionamento. Em termos de hierarquia, um nó não pode ser seu pai, apesar de poder ser o pai de outros nós.

Ou seja, em aplicações nas quais a modelagem requeira um auto relacionamento, o banco de dados baseado em grafos não deve ser escolhido como solução. Outra desvantagem diz respeito à capacidade de replicação que, apesar de muito boa, no Neo4J só podem ser replicados grafos inteiros.

O Neo4J lidera o ranking de popularidade dos SGBDs em [grafos](#). Outros exemplos de bancos de dados baseados em grafos são: [Microsoft Azure Cosmos](#), [OrientDB](#), [ArangoDB](#), [Virtuoso](#) e [Amazon Neptune](#) entre outros.

A seguir, apresentaremos algumas características específicas do Neo4J, a saber:

1

Controle de resiliência ou tolerância a falhas - o Neo4J tem suporte para a execução de *backups* "frios"¹ e, também, no caso dos backups denominados "quentes"².

2

Tem recursos para uso em *clusters* de alta disponibilidade.

3

Possui uma linguagem de consulta denominada Cypher, que será usada na próxima seção.

Comandos NOSQL para a definição da estrutura do banco de dados em grafos

O Neo4J tem uma linguagem declarativa chamada Cypher, projetada especificamente para consultar grafos e seus componentes. Os comandos da linguagem são pouco semelhantes à linguagem SQL e são destinados a consultas aos dados em grafos.

O uso dessa linguagem e a geração do grafo resultante permitem uma melhor visualização ou interpretação da consulta por parte do usuário, fornecendo uma maneira familiar e legível para corresponder a padrões de nós e arestas dentro de um conjunto de dados em grafo.

Assim como a SQL, é uma linguagem declarativa que permite indicar os comandos a serem executados, tal como *match*, *insert*, *update* ou *delete*.

O ambiente gráfico do Neo4J é rico, mas compreender a linguagem de comandos torna o profissional mais independente de interfaces e conhecedor dos detalhes do banco de dados.

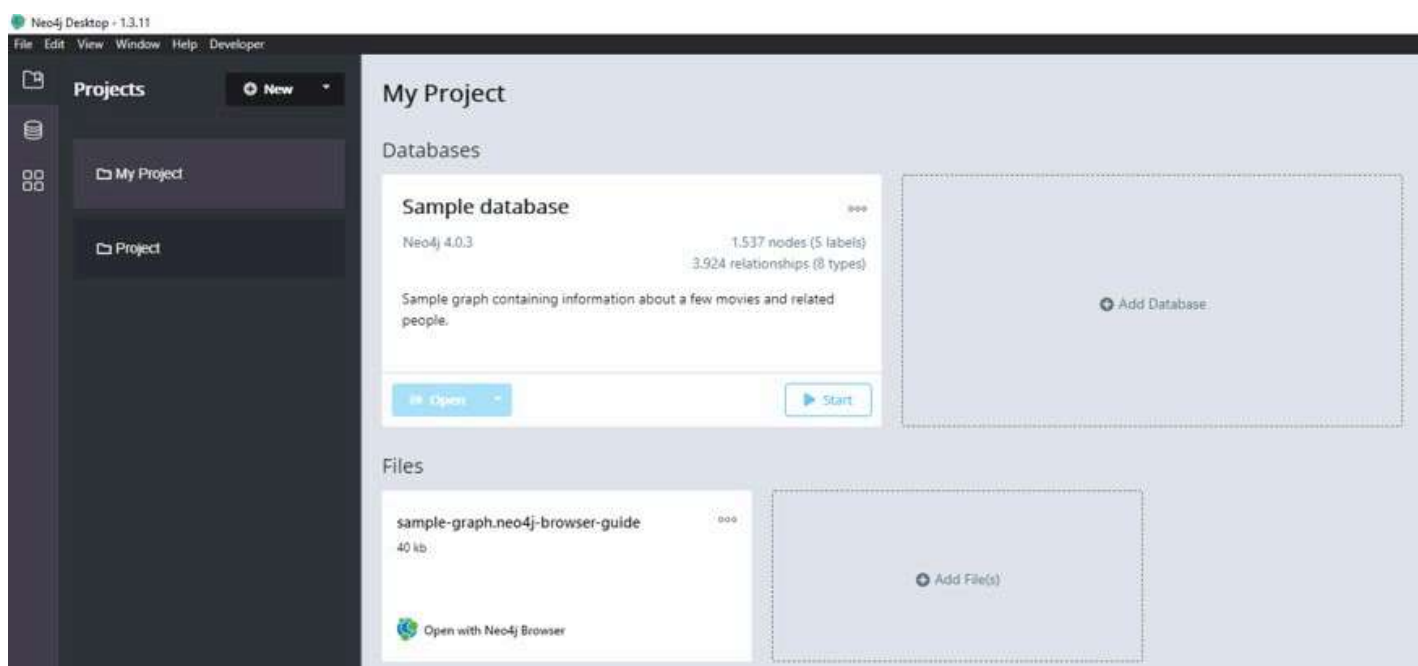
1

Para iniciar o estudo, vamos criar um banco de dados e, para tal, considerando que você já tenha o Neo4J instalado, clique em Add Database (Figura 2) e escolha a opção Create a Local Database. Então, você pode definir o nome do *database*. No exemplo, foi definido o nome Aula e com a senha "admin".

2

Clique em Create e, depois que criar o *database*, clique em Start. Observe a descrição indicando zero *nodes* (nós) e *relationships* (relacionamentos). Clique em seguida em Open. Na tela que abrir, tem algumas opções para facilitar o primeiro contato com o Neo4J indicando o "Getting started with Neo4J" ou "Try Neo4J with live data" ou ainda o "Cypher basics".

Figura 2 - Tela de abertura do Neo4J com a opção de adicionar um banco de dados.



Fonte: Neo4J.

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Observe o cursor no alto da tela (Figura 3); nesse local copie e cole o código abaixo para a criação de nosso primeiro exemplo. Clique no botão Run ou no atalho CTRL + Enter:

```
CREATE (ana:Pessoa {nome: 'Ana' , email: 'ana@meumail.com'})

CREATE (lia:Pessoa {nome: 'Lia' , email: 'lia@meumail.com'})

CREATE (luis:Pessoa {nome: 'Luís' , email: 'luis@meumail.com'})

CREATE (rui:Pessoa {nome: 'Rui' , email: 'rui@meumail.com'})

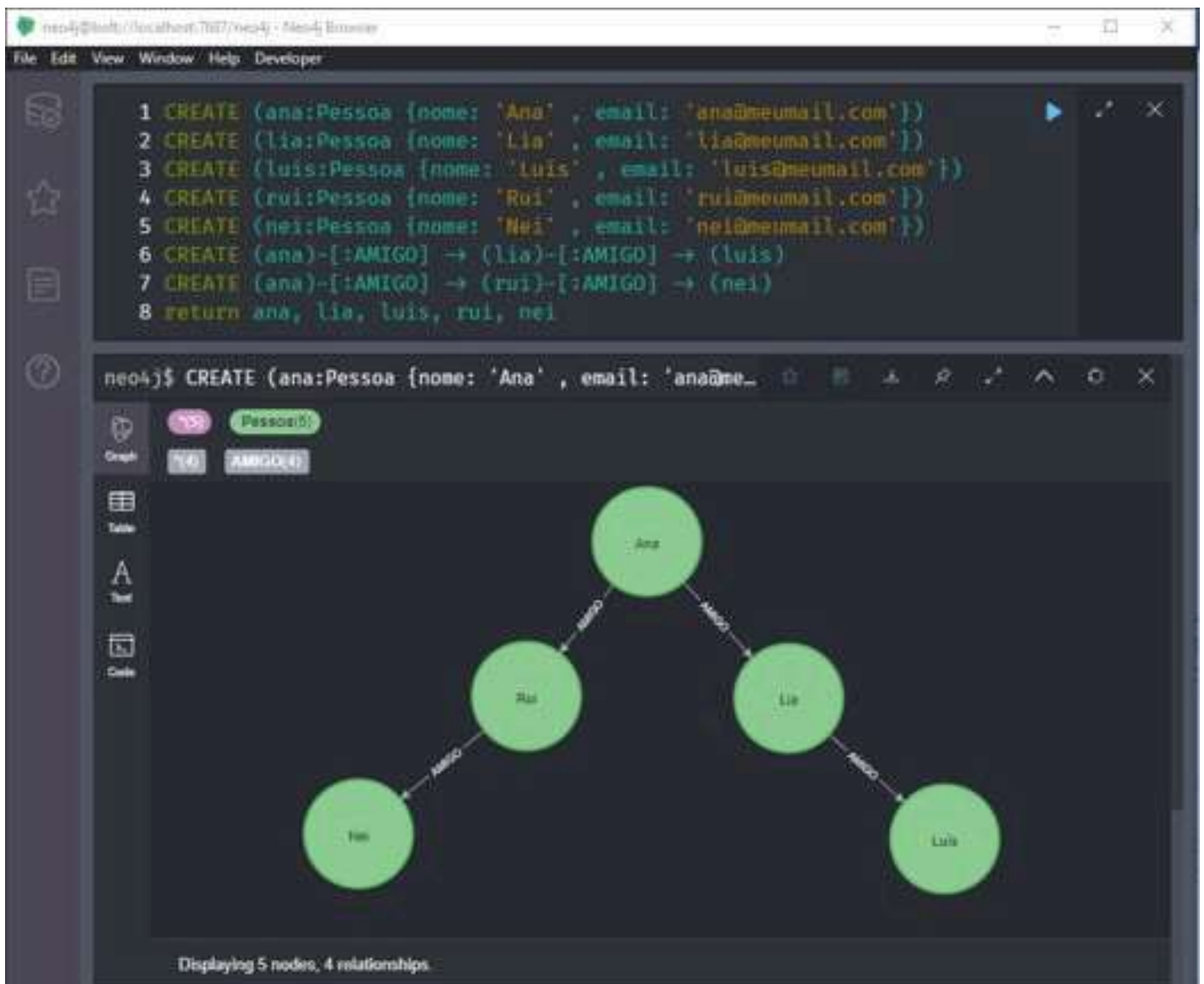
CREATE (nei:Pessoa {nome: 'Nei' , email: 'nei@meumail.com'})

CREATE (ana)-[:AMIGO] -> (lia)-[:AMIGO] -> (luis)

CREATE (ana)-[:AMIGO] -> (rui)-[:AMIGO] -> (nei)
```

```
return ana, lia, luis, rui, nei
```

Figura 3 - Resultado da execução do comando para criação de 5 nós e 4 relacionamentos.



Fonte: Neo4J.

O comando CREATE define o nó e suas propriedades. Logo após a criação dos nós, podemos criar os relacionamentos usando também o CREATE e indicando esses tipos de relacionamentos entre colchetes. No comando, foram criados cinco nós indicando pessoas que possuem um relacionamento de amizade entre elas.

Cada CREATE representou uma pessoa e para cada uma delas foram armazenados no banco dois pares de chaves-valor com as propriedades do nome e do e-mail delas. Logo após a criação dos nós, foram criados os relacionamentos, com a indicação pela seta do sentido do relacionamento. Assim, Ana tem amizade com Lia, que por sua vez tem amizade com Luís.

Comentário

Você pode movimentar os nós à vontade e, mesmo assim, as setas irão manter os sentidos pré-definidos na criação dos relacionamentos entre os nós.

Observe também que, na parte esquerda da tela, existem algumas opções de visualização, um dos pontos fortes do Neo4J, ou seja, a interface gráfica e seus recursos. Na ordem das opções, é apresentada a opção de visualização com os grafos; abaixo, a opção de visualização dos dados em forma de tabela (chave-valor indentada), em formato texto e, por fim, apresentando o código-fonte.

Os dados dos nós podem ser visualizados pelo comando abaixo, que é comparado a um `SELECT * FROM <nome_da_tabela>`. Vejamos:

```
MATCH (n) RETURN n
```

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Caso fosse necessário especificar alguma propriedade, poderia essa ser definida como, por exemplo, no comando a seguir; retorna apenas os nomes das pessoas:

```
MATCH (n) RETURN n.nome
```

Para a consulta anterior listar os nomes por ordem alfabética, basta acrescentar um `ORDER BY`. Vamos aproveitar e mostrar como apresentar mais de uma propriedade: No caso, listando o nome e o *e-mail* das pessoas. Vejamos:

```
MATCH (n) RETURN n.nome, n.email ORDER BY n.nome
```

Na consulta a seguir, iremos apresentar um exemplo de comando para a alteração do valor de uma propriedade, algo semelhante ao que é realizado com a aplicação de um comando `UPDATE` em SQL. Vejamos o comando para alterar o valor do nome da pessoa Lia para Lia Maria:

```
MATCH ( n { nome : 'Lia' } )  
  
SET n.nome = 'Lia Maria'  
  
RETURN n.nome
```

Para confirmar a alteração, podemos executar o comando a seguir para listar todos os dados ou propriedades da pessoa Lia Maria. Lembre-se que o `MATCH` representa o `WHERE` no relacional. Vejamos então o código para tal:

```
MATCH ( n { nome : 'Lia Maria' } )
```

```
RETURN n
```

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Atividade

1. Sobre as características do modelo de dados baseado em grafos, podemos afirmar em relação à sua estrutura, exceto:

- a) No modelo em grafos os nós representam os registros que são usados no banco de dados relacional.
- b) No modelo em grafos os nós podem ter vários relacionamentos para com outros nós.
- c) Os nós têm propriedades que são comparadas aos atributos no modelo relacional.
- d) Os nós e relações suportam propriedades, um par de valores-chave em que os dados são armazenados.
- e) Os relacionamentos apresentam sentido bidirecional, um tipo, um nó inicial e um nó final.

2. Assinale a afirmativa incorreta quanto às vantagens e tipos de aplicações indicadas para uso do banco de dados em grafos:

- a) O modelo de dados em grafos facilita a navegação pelos relacionamentos, algo que não é possível em SGBDs relacionais devido às estruturas de tabelas rígidas e à impossibilidade de seguir conexões entre os dados..
- b) É indicado para o uso em aplicações transacionais com o uso da integridade referencial.
- c) O banco de dados baseado em grafos é indicado para aplicações científicas e técnicas ou ainda para o tratamento de aplicações complexas.
- d) Como exemplos de aplicações indicadas para serem desenvolvidas com esse modelo de dados podem ser listadas as aplicações para o gerenciamento de dados geográficos ou para modelar redes de computadores de um provedor de telecomunicações.
- e) O banco de dados em grafo é indicado para aplicações que têm no cruzamento de dados distintos uma possibilidade de melhor análise.

3. Considere o código a seguir e indique a afirmativa correta a respeito do que ele faz:

```
CREATE ( ana : gerente { matricula : 100, nome : 'Ana' } )
CREATE ( luis : gerente { matricula : 101, nome : 'Luís' } )
CREATE ( lia : funcionario { matricula : 102, nome : 'Lia' } )
CREATE ( rui : funcionario { matricula : 103, nome : 'Rui' } )
CREATE ( nei : funcionario { matricula : 104, nome : 'Nei' } )
CREATE ( projeto_1 : projeto { nrprojeto : 1, nome : 'Projeto 1' } )
CREATE ( projeto_2 : projeto { nrprojeto : 2, nome : 'Projeto 2' } )
CREATE ( ana )-[:GERENTE]-> ( projeto_1 )
CREATE ( luis )-[:GERENTE]-> ( projeto_2 )
CREATE ( lia )-[:TRABALHA]-> ( projeto_1 )
CREATE ( rui )-[:TRABALHA]-> ( projeto_1 )
CREATE ( nei )-[:TRABALHA]-> ( projeto_2 )
```

- a) Cria 7 nós representando os relacionamentos de trabalho entre eles, mas com apenas um tipo de relacionamento entre todos os nós.
 - b) Cria 12 nós representando os relacionamentos de trabalho entre eles, com apenas 2 tipos de relacionamentos entre todos os nós.
 - c) Cria 7 nós representando os relacionamentos de trabalho entre eles, com 3 tipos de nós e 2 tipos de relacionamentos.
 - d) Cria 5 nós representando os relacionamentos de trabalho entre eles, com 3 tipos de nós e 2 tipos de relacionamentos.
 - e) Cria 12 nós representando os relacionamentos de trabalho entre eles, com 2 tipos de nós e 3 tipos de relacionamentos.
-

Notas

Backup frio¹

Quando o banco de dados não está em execução.

Backup quente²

Quando em execução.

Referências

ELMASRI, R.; NAVATHE, S.B. **Sistemas de banco de dados**. 6.ed. São Paulo: Pearson Education do Brasil, 2018.

SADALAGE, P.J.; FOWLER, M. **NoSQL essencial** – um guia conciso para o mundo emergente da persistência poliglota. 1.ed. São Paulo: Novatec, 2013.

Próxima aula

- Comandos Cypher para manipulação de dados: insert, update e delete;
- Comandos Cypher para consultas simples e com o uso de funções;
- Comandos Cypher para consultas envolvendo agregações.

Explore mais

- Leia o livro *Sistema de banco de dados*
- Leia o capítulo 11 do livro *NoSQL Essencial - Um Guia Conciso para o Mundo Emergente da Persistência Poliglota*