

Neo4J

Prof. Igor Avila Pereira
igor.pereira@riogrande.ifrs.edu.br

Divisão de Computação
Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
Câmpus Rio Grande

Agenda I

1 Introdução

2 Neo4J

- Introdução
- Instalação
- Nomeclaturas
- Primeiros comandos

3 Client Java

Agenda

- 1 Introdução
- 2 Neo4J
- 3 Client Java

Introdução

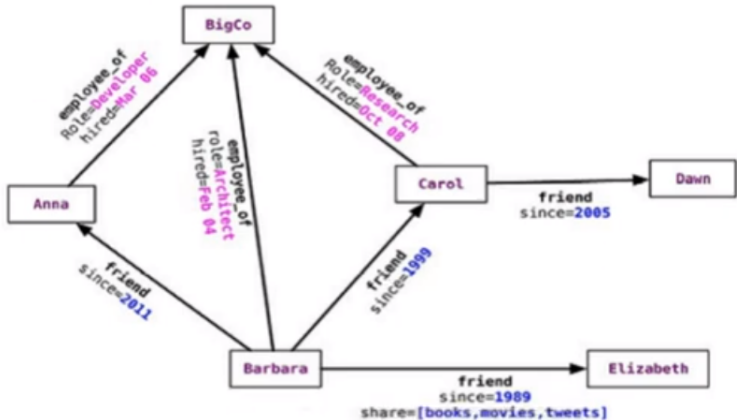
- Os bancos de dados baseados em grafos permitem que além das entidades, sejam também armazenados os relacionamentos entre elas
- Os elementos básicos de um grafo são:
 - nó - instância de um objeto
 - aresta - relacionamento (pode ter propriedades)
 - Obs: Aqui, as arestas são sempre direcionadas e existe um nó inicial e final.

Introdução

- Podemos acrescentar mais propriedades às arestas, por exemplo, em um relacionamento de Amizade podemos adicionar uma data
- As consultas em um banco de grafos são chamadas de percurso
 - Os percursos não alteram os nós
- Percorrer um grafo é mais rápido que realizar junções em banco de dados relacionais



Introdução



Agenda

1 Introdução

2 Neo4J

- Introdução
- Instalação
- Nomeclaturas
- Primeiros comandos

3 Client Java

Introdução

- O Neo4J é um banco de dados não relacional (NoSQL), orientado à grafos
- **Site Oficial:** neo4j.com
- Tem 2 versões: **Community** e Enterprise
- O Neo4J foi o primeiro banco de grafo, possui uma comunidade ativa de mais de 20 mil membros;
- É um banco de fácil utilização;
- Permite que além das entidades (nós), sejam armazenados também os relacionamentos (arestas) entre elas;
- No Neo4J, como na maioria de banco de dados orientado a grafos, as arestas são sempre direcionadas:
 - Nó inicial (de origem) e
 - Nó final (de destino)

Introdução

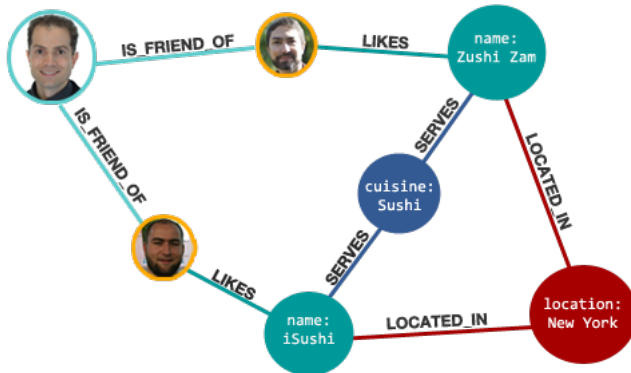
- Os dados são consultados mediante o percurso pelos dados armazenados no grafo
- Tanto os nós como arestas podem ter propriedades
- Percorrer grafos é mais rápido que realizar junções nos bancos de dados relacionais
- O Neo4J tem como prioridade, tratar seus relacionamentos da melhor forma possível, isso quer dizer que:
 - Na medida que os relacionamentos entre “nós” aumentam, sua capacidade de processamento continua estável, diferentemente, de bancos de dados relacionais tradicionais.

Introdução

- A linguagem do Neo4J é a *cypherQuery*, e a curva de aprendizagem não é tão grande
- O Neo4j permite que a partir de um nó, sejam retornados todos os relacionamentos (ou podemos optar pela direção)
 - INCOMING
 - OUTGOING
- todas as consultas partem de um nó inicial
- Podemos percorrer até um certo nível de profundidade ou estabelecer padrões para serem buscados

Introdução

- Aqui está um exemplo de como encontrar um restaurante japonês localizado em New York, baseado no gosto dos meus amigos:



Introdução

- Devemos modelar com cuidado: Adicionar nós é fácil, mas alterar é complexo
- Podemos também aplicar os algoritmos de grafos nos bancos: Dijkstra, Min-Max, dentre outros;
- Os bancos de grafos geralmente rodam em uma única máquina
 - **Exceções:** Infinite Graph
- Os nós são consistentes:
 - O Neo4j é compatível com ACID
 - Atomicidade;
 - Consistência;
 - Isolamento;
 - Durabilidade.
- A maioria dos bancos de dados em grafo garantem consistência através de transações

Introdução

Escalabilidade

- Em bancos de grafos a fragmentação é difícil, por não ser orientada a *clusters*
- Caso seja utilizado um banco distribuído, é interessante que os relacionamentos de um nó estejam em uma mesma máquina

Onde Usar

- Qualquer domínio rico em relacionamentos é apropriado. Mesmo que esses relacionamentos ocorram em diferentes domínios. Ex: detecção de fraudes, busca baseada em grafos, redes sociais, sistemas de recomendação, dentre outros e etc.
- <https://neo4j.com/use-cases/>

Instalação

- **Ubuntu/Linux Mint:**

<https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-neo4j-on-ubuntu-20-04>

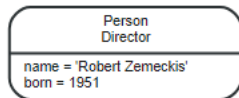
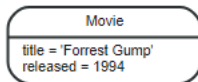
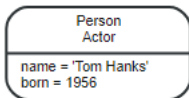
- **Docker:**

```
-- 1 vez
sudo docker run \
--publish=7474:7474 --publish=7687:7687 \
--volume=$HOME/neo4j/data:/data \
--env=NEO4J_AUTH=neo4j/password \
--name neo4j neo4j
-- demais vezes
sudo docker start neo4j
sudo docker exec -it neo4j bash
```

- A interface estará disponível em <http://localhost:7474>
- **login:** *neo4j* / **senha:** *password*

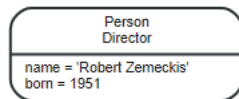
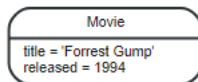
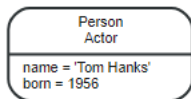
Nomeclaturas

- Antes de iniciar de fato, é preciso explicar as nomeclaturas do Neo4J, irei dar uma breve descrição:
 - **Labels:** É uma espécie de template para os nós.
 - A grosso modo, seria uma classe. Na imagem abaixo, temos as *labels*: **PersonActor**, **Movie** e **PersonDirector**.



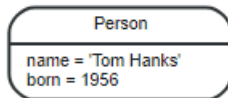
Nomeclatura

- **Propriedades (Properties):** são parâmetros baseados em chave-valor, para designar um nó, ou uma relacionamento.
- Na imagem há varias propriedades, tais como: **name**, **born**, **title** e **released**, eles demonstram características ou descrições de uma *label*.



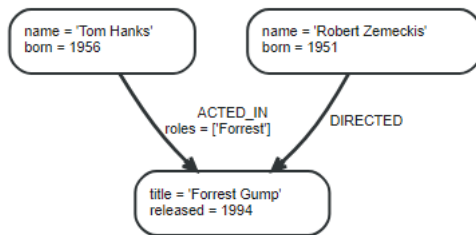
Nomeclatura

- **Nós(Nodes):** São como entidades baseadas em um template pronto (label), se parecem instâncias de objetos de uma classe.



Nomeclatura

- **Relacionamentos:** Representa uma conexão entre nós, é possível criar propriedades dentro de um relacionamento.
- No exemplo abaixo, há dois relacionamentos, o primeiro se chama `ACTED_IN`, que possui uma propriedade chamada `roles`, o segundo relacionamento se chama `DIRECTED`, que não possui propriedades.



Primeiros Comandos

- A linguagem *Cypher* possui a seguinte sintaxe:
 - START beginNode = (nó inicial)
 - MATCH (relacionamento)
 - WHERE (condição de filtragem)
 - RETURN (o que retornar: nós, relacionamentos, propriedades)
 - ORDER BY (propriedades para ordenar)
 - SKIP (nodos para ignorar)
 - LIMIT (limitar os resultados)

Primeiros Comandos

Agora o comando CREATE:

```
1 CREATE (p:Pessoa {nome: 'Marlon Alves', profissão:'Desenvolvedor de sistemas'})
2 CREATE (c:Cidade {nome: 'Santos'})
3 CREATE (p)-[:RESIDE_EM]->(c)
4 return *
```

\$ CREATE (p:Pessoa {nome: 'Marlon Alves', profissão:'Desenvolvedor de s...}

Graph

* (2) Cidade(1) Pessoa(1)

* (1) RESIDE_EM(1)

Table

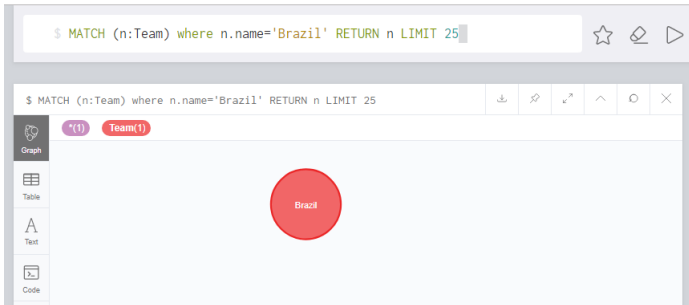
Text

Code

```
graph LR;
  Marlon((Marlon Alves)) -- RESIDE_EM --> Santos((Santos))
```

Primeiros comandos

- No exemplo abaixo, a cláusula MATCH retorna um nó, nesse caso sendo filtrado pela condição do WHERE (propriedade nome do nó Brazil dentro da label Team), com limite de 25 registros.



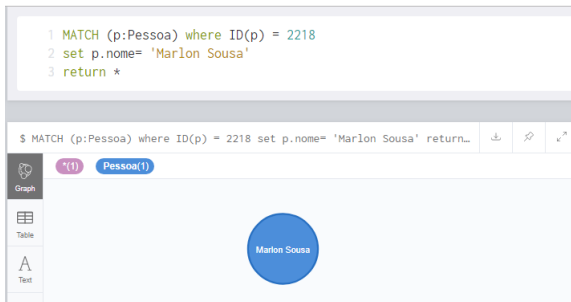
Primeiros Comandos

- Na imagem, foi criado um nó **Pessoa**, com propriedade nome: **Marlon Alves** e profissão **Desenvolvedor de sistemas**
- Na linha foi criado um nó **Cidade** com o nome: **Santos**, e por último há um relacionamento entre **p(Pessoa)** e **c(cidade)**
- Nota-se que **p** e **c** são *alias* de seus respectivos nós, o símbolo de apontamento no relacionamento indica que **Marlon Alves** reside em **Santos**.

Primeiros Comandos

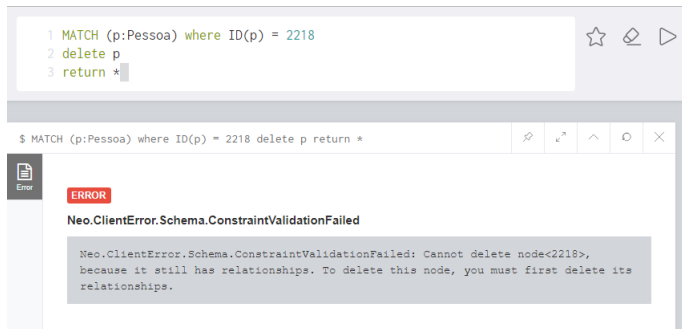
Atualizando e deletando um nó. . .

- Para alterar uma propriedade, é usado o comando abaixo
- Nesse caso, filtramos o nó Pessoa pelo seu id único criado automaticamente pelo Neo4J.



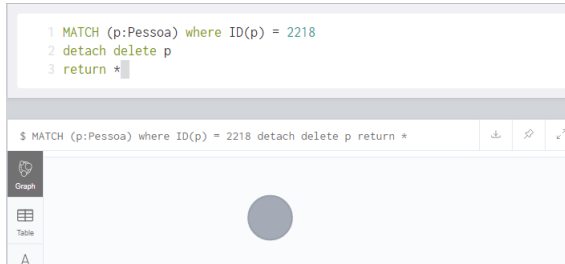
Principais Comandos

- Para deletar é usado o comando **delete**, porém nota-se que nesse caso houve um erro na execução.
- Isso ocorreu porque há relacionamento entre o nó Pessoa escolhido.



Principais Comandos

- Nesse caso, é necessário usar o comando DETACH DELETE, que irá excluir o nó e seu relacionamento com o outro Nó.



Agenda

- 1 Introdução
- 2 Neo4J
- 3 Client Java

Client Java

- No Neo4j para criar um nó basta utilizar comandos simples e setar as suas propriedades.

```
Node martin = graphDB.createNode();    Node pramod= graphDB.createNode();  
martin.setProperty("name", "Martin");  pramod.setProperty("name", "Pramod");
```

- Após isso, podemos criar os relacionamentos

```
Martin.createRelationship(pramod, FRIEND);  
Pramod.createRelationship(martin, FRIEND);
```

- Obs: Martin e Pramod deveriam estar em minúsculo

Neo4J

Prof. Igor Avila Pereira
igor.pereira@riogrande.ifrs.edu.br

Divisão de Computação
Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
Câmpus Rio Grande