

# 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

### 1 Introdução

### 2 Neo4J

- Introdução
- Instalação
- Nomeclaturas

### 3 Comandos *Cypher*

- CREATE
- MATCH (Pesquisa)
- MATCH (Atualização)
- MATCH (Adicionando/Removendo Propriedades)
- DELETE/DETACH DELETE

### 4 Client Java

- Instalação
- Conexão
- CREATE
- MATCH
- DELETE/DETACH DELETE

# Agenda

- 1 Introdução
- 2 Neo4J
- 3 Comandos *Cypher*
- 4 Client Java

## Introdução

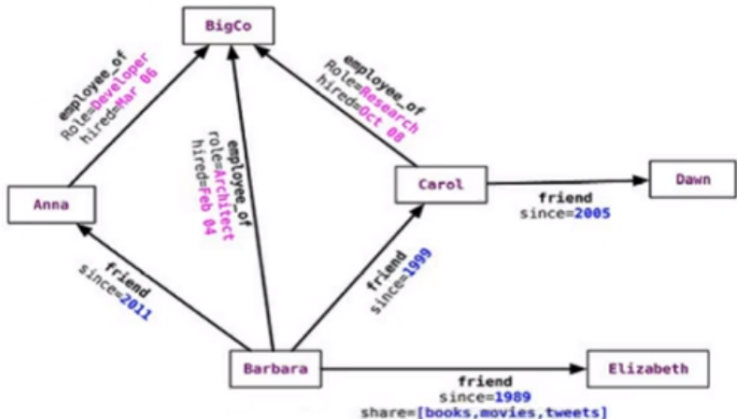
- Os bancos de dados baseados em grafos permitem que além das entidades, sejam também armazenados os relacionamentos entre elas
- Os elementos básicos de um grafo são:
  - nó - instância de um objeto
  - aresta - relacionamento (pode ter propriedades)
  - Obs: Aqui, as arestas são sempre direcionadas 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
- 3 Comandos *Cypher*
- 4 Client Java

## Introdução

- O Neo4J é um banco de dados não relacional (NoSQL), orientado à grafos
- **Site Oficial:** [neo4j.com](http://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

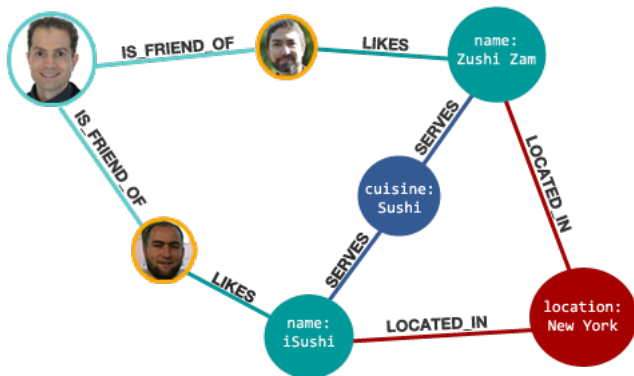
- Os dados são consultados mediante o percurso pelos dados armazenados no grafo
- Tanto os nós como arestas podem ter propriedades
- Percorrer grafos é mais rápido que realizar junções nos bancos de dados relacionais
- O Neo4J tem como prioridade, tratar seus relacionamentos da melhor forma possível, 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

### Docker: 1ª vez:

```
1 sudo docker run \  
2 --publish=7474:7474 --publish=7687:7687 \  
3 --volume=$HOME/neo4j/data:/data \  
4 --env=NEO4J_AUTH=neo4j/password \  
5 --name neo4j neo4j
```

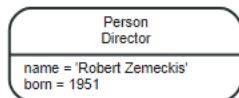
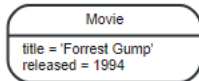
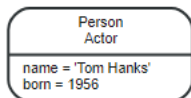
### Demais vezes

```
1 sudo docker start neo4j  
2 sudo docker exec -it neo4j bash
```

- A interface do Neo4j estará disponível em:
  - <http://localhost:7474>
- **login:** *neo4j* e **senha:** *password*

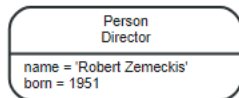
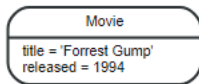
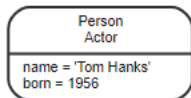
## Nomeclaturas

- Antes de iniciar de fato, é preciso explicar as nomenclaturas 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**.



## Nomenclatura

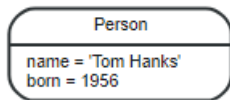
- **Propriedades (Properties):** são parâmetros baseados em chave-valor, para designar um nó, ou uma relacionamento.
- Na imagem há varias propriedades, tais como: **name**, **born**, **title** e **released**, eles demonstram características ou descrições de uma *label*.





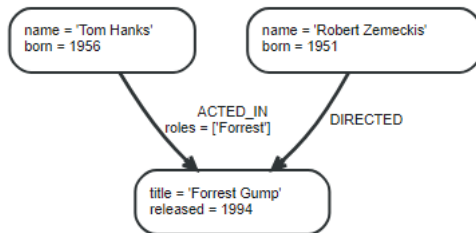
## Nomenclatura

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



## Nomenclatura

- **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.



## Agenda

### 1 Introdução

### 2 Neo4J

### 3 Comandos *Cypher*

- CREATE
- MATCH (Pesquisa)
- MATCH (Atualização)
- MATCH (Adicionando/Removendo Propriedades)
- DELETE/DETACH DELETE

### 4 Client Java

## Comandos *Cypher*

- A linguagem *Cypher* permite executar *queries* complexas escrevendo pouco
- É intuitiva (parecida com SQL)
- 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)

## CREATE

### 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

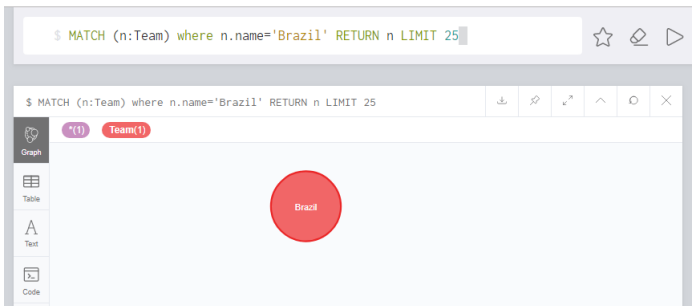
```
graph LR;
  MarlonAlves((Marlon Alves)) -- RESIDE_EM --> Santos((Santos));
```

## CREATE

- Na Figura do slide anterior, foi criado um nó **Pessoa**, com propriedade nome: **Marlon Alves** e profissão **Desenvolvedor de sistemas**
- Também foi criado um nó **Cidade** com o nome: **Santos**, e por último há um relacionamento entre **p(Pessoa)** e **c(cidade)**
- **Obs:** 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**.

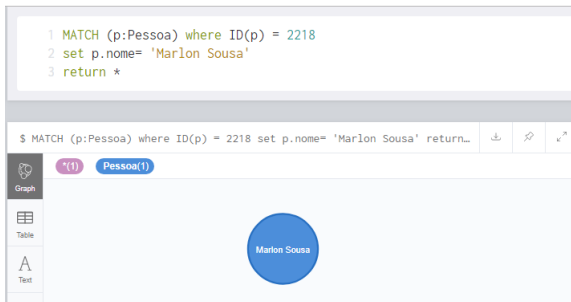
## MATCH (Pesquisa)

- 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.



## MATCH (Atualização)

- Para alterar uma propriedade, é usado o comando abaixo:



- Nesse caso, filtramos o nó Pessoa pelo seu id único criado automaticamente pelo Neo4J.



## MATCH (Adicionando/Removendo Propriedades)

### Adicionando Propriedade:

```
1 MATCH (n:Pessoa) WHERE n.nome = "Igor" SET n += {profissao: "professor"}
```

- O nó **Igor** terá uma nova propriedade (**profissao**) com valor **professor**

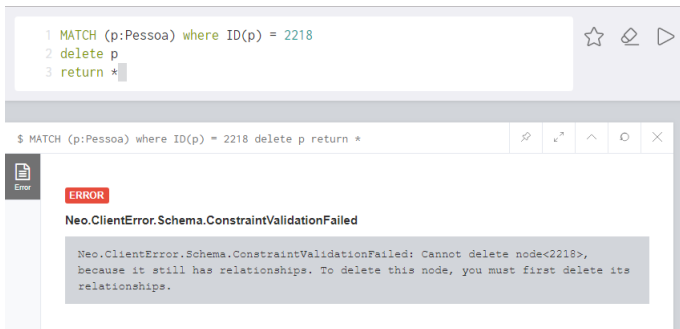
### Removendo Propriedade:

```
1 MATCH (n:Pessoa) WHERE n.nome = "Igor" SET n.profissao = null
```

- O nó **Igor** não terá mais a propriedade (**profissao**)

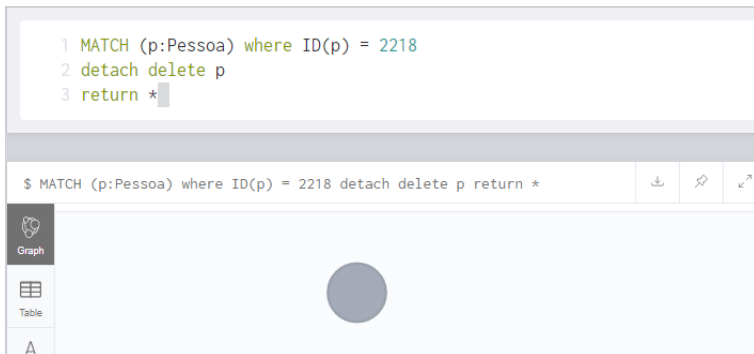
## DELETE/DETACH DELETE

- 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.



## DELETE/DETACH DELETE

- 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 Comandos *Cypher*

4 Client Java

- Instalação
- Conexão
- CREATE
- MATCH
- DELETE/DETACH DELETE

## Instalação

### Maven:

```
1 <dependencies>
2   <dependency>
3     <groupId>org.neo4j.driver</groupId>
4     <artifactId>neo4j-java-driver</artifactId>
5     <version>4.4.0</version>
6   </dependency>
7 </dependencies>
```

## Conexão

```
1 Driver driver = GraphDatabase.driver("bolt://localhost:7687",  
2   AuthTokens.basic("neo4j", "password"));
```

- **Obs:**

- usuário: *neo4j*
- senha: *password*

**Link:** [Get-started](#)

## CREATE

### Criando um nodo:

```
1 Pessoa p1 = new Pessoa("111.111.111-11", "Igor",  
2     LocalDate.of(1987, 01, 20));  
3  
4 try (Session session = driver.session()) {  
5  
6     session.run("CREATE(p:Pessoa{cpf:$cpf,  
7         nome:$nome,nascimento:$nascimento})",  
8         parameters("cpf", p1.getCpf(),  
9             "nome", p1.getNome(),  
10            "nascimento", p1.getNascimento()));  
11 } finally {  
12     driver.close();  
13 }
```

## CREATE

### Criando um relacionamento entre o nodo p1 e p2:

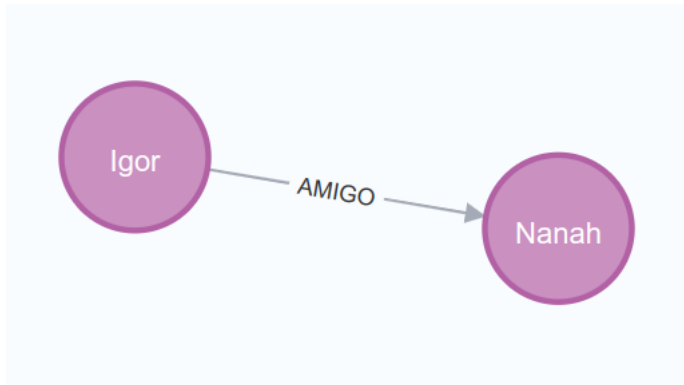
```
1 try (Session session = driver.session()) {  
2  
3     session.run("MATCH (p1: Pessoa{cpf: $cpf}), (p2:Pessoa{cpf:$cpf2})  
4         CREATE (p1)-[:AMIGO]->(p2)", parameters("cpf", p1.getCpf(),  
5             "cpf2", p2.getCpf()));  
6  
7 } finally {  
8     driver.close();  
9 }
```

**Obs:** É obrigatório criar os nodos p1 e p2 anteriormente



## CREATE

### Resultado:



## MATCH

```
1 try (Session session = driver.session()) {
2
3     Result result = session.run("MATCH (p: Pessoa {cpf:$cpf}) RETURN
4         p.nome", parameters("cpf", p1.getCpf()));
5
6     while(result.hasNext()){
7         System.out.println(result.next().get("p.nome"));
8         // funciona tb desta forma
9         // System.out.println(result.next().get(0));
10    }
11 } finally {
12     driver.close();
13 }
```

## MATCH

```
1 try (Session session = driver.session()) {  
2  
3     Result result2 = session.run("MATCH (p: Pessoa {cpf:$cpf})  
4     RETURN p", parameters("cpf", pessoa1.getCpf()));  
5  
6     while(result2.hasNext()){  
7         System.out.println(result2.next().get("p").get("nome"));  
8     }  
9  
10 } finally {  
11     driver.close();  
12 }
```

## DELETE/DETACH DELETE

### Deletando todos os nós:

```
1 try (Session session = driver.session()) {  
2     session.run("MATCH (n) DETACH DELETE n");  
3 } finally {  
4     driver.close();  
5 }
```

# 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