

## Neo4J - Cypher

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 Neo4j

- Instalação

### 2 Cypher

- Tipos
- Convenção de Nomes
- CREATE
- MATCH (Pesquisa)
- MATCH (Atualização)
- DELETE/DETACH DELETE

### 3 Client Java

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

## Agenda

- 1 Neo4j
  - Instalação
- 2 *Cypher*
- 3 Client Java

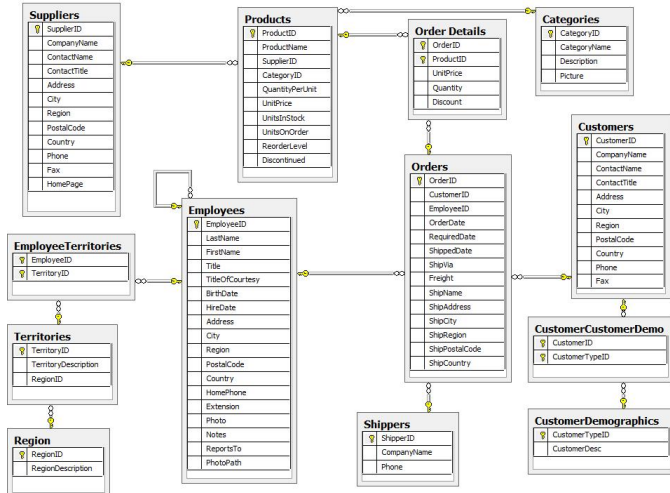
- Os bancos de dados baseados em grafos permitem que além das entidades, sejam também armazenados os relacionamentos entre elas
- Os elementos básicos de um grafo são:
  - nó - instância de um objeto
  - aresta - relacionamento (pode ter propriedades)
  - Obs: Aqui, as arestas são sempre direcionadas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

## Neo4j

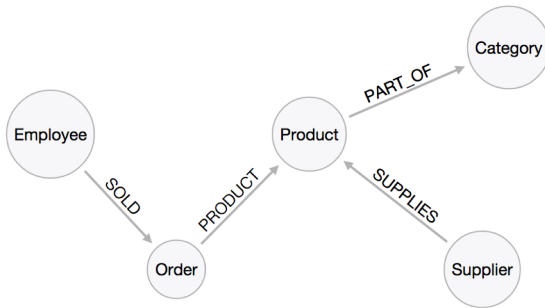
- Podemos acrescentar mais propriedades às arestas, por exemplo, em um relacionamento de Amizade podemos adicionar uma data
- As consultas em um banco de grafos são chamadas de percurso
  - Os percursos não alteram os nós
- Percorrer um grafo é mais rápido que realizar junções em banco de dados relacionais



# Neo4j



## Neo4j



## Instalação

### Docker:

```
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*



## Agenda

### 1 Neo4j

### 2 *Cypher*

- Tipos
- Convenção de Nomes
- CREATE
- MATCH (Pesquisa)
- MATCH (Atualização)
- DELETE/DETACH DELETE

### 3 Client Java

## 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)

## Tipos de Propriedades

- ① **Number**, um tipo abstrato, com os subtipos:
  - Integer
  - Float
- ② **String**
- ③ **Boolean**
- ④ **Point** (tipo espacial)
- ⑤ Tipos temporais:
  - Date
  - Time
  - LocalTime
  - DateTime
  - LocalDateTime
  - Duration

## Tipos Estruturais

### 1 Node (Nodo)

- id
- label
- Map de propriedades

### 2 Relationship (Relacionamento)

- Id
- Type
- Map de propriedades
- Id do nodo de início
- Id do nodo de fim

## Convenção de Nomes

- **Caracteres Alfanuméricos:**
  - Nomes devem começar com algum caracter alfanumérico
  - Isto inclui caracteres '*non-English*' como å, ä, ö, ü etc.
- **Números:**
  - Nomes não devem começar com um número
- **Símbolos:**
  - Nomes não devem conter símbolos, exceto *underscore* ou \$ para denotar um parâmetro
- **Tamanho:**
  - Pode ser muito longo, até 65535 ( $2^{16} - 1$ ) ou 65534 caracteres, dependendo da versão do Neo4j

## Convenção de Nomes

- **Case-sensitive:**

- Os nomes diferenciam maiúsculas de minúsculas e, portanto, :PERSON, :Persone :person são três rótulos diferentes e *n* e *N* são duas variáveis diferentes.

- **Caracteres de espaço em branco:**

- Os caracteres de espaço em branco à esquerda e à direita serão removidos automaticamente.
- Por exemplo, MATCH (a) RETURN a é equivalente a MATCH (a) RETURN a.

## Convenção de Nomes

- Labels de nodos, tipos e propriedades de relacionamentos podem ter nomes reutilizados. Ex:

```
1 CREATE (a:a {a: 'a'})-[r:a]->(b:a {a: 'a'})
```

- Variáveis para nodos e relacionamentos não devem ter nomes reutilizados no mesmo escopo de consulta.
  - Ex: consulta inválida já que o nodo e o relacionamento possuem o mesmo nome **a**:

```
1 CREATE (a)-[a]->(b).
```

## Recomendações

- **Node labels:**

- Camel-case, começando com um caracter maiúsculo
- Ex: VehicleOwner

- **Relationship types:**

- Caracteres maiúsculo, usando *underscore* para separar palavras
- Ex: :OWNS\_VEHICLE



## CREATE

### Criando Nodos:

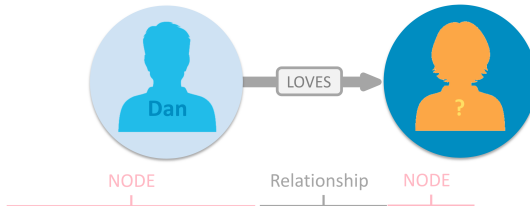
```
1 CREATE(p:Pessoa{cpf: "111.111.111-11", nome: "joao", idade:20})
2 return p
```

### Criando Arestas:

```
1 -- exemplo 1
2 MATCH (p1:Pessoa), (p2:Pessoa)
3 where p1.nome = "João" and p2.nome = "Maria"
4 CREATE (p1)-[:AMIGO]->(p2)
5
6 -- exemplo 2
7 MATCH (p1:Pessoa), (p2:Pessoa)
8 where ID(p1) = 10 and ID(p2) = 11
9 CREATE (p1)-[:AMIGO]->(p2)
```

## MATCH (Pesquisa)

Quais são as pessoas que amam Dan?



**MATCH** (:Person { name:"Dan" } ) -[:LOVES]-> ( whom ) **RETURN** whom

LABEL      PROPERTY      VARIABLE

## MATCH (Pesquisa)

### Operadores:

- **Matemáticos:**

1 +, -, \*, /, % e ^

- **Comparação:**

1 =, <>, <, >, <=, >=, IS NULL, IS NOT NULL

- **Strings:**

1 STARTS WITH, ENDS WITH, CONTAINS, =~ para regex matching

- **Concatenação de Strings:**

1 +

## MATCH (Pesquisa)

### Operadores:

- **DISTINCT:**

```
1  -- remove valores duplicados
2  DISTINCT
```

- **Booleano:**

```
1  AND, OR, XOR, NOT
```

**Obs:** mais exemplos de consultas com operadores podem ser encontrados [aqui](#)

## MATCH (Pesquisa)

Uma consulta SQL como esta abaixo:

```
1 SELECT p.* FROM products as p;
```

Ficaria assim escrita em *Cypher*:

```
1 MATCH (p:Product) RETURN p;
```

## MATCH (Pesquisa)

### SQL:

```
1 SELECT p.ProductName, p.UnitPrice FROM products as p
2     ORDER BY p.UnitPrice DESC LIMIT 10;
```

### *Cypher:*

```
1 MATCH (p:Product)
2 RETURN p.productName, p.unitPrice
3 ORDER BY p.unitPrice DESC
4 LIMIT 10;
```

## MATCH (Pesquisa)

### Pular os 3 primeiros resultados:

```
1 MATCH (n)
2 RETURN n.name
3 ORDER BY n.name
4 SKIP 3
```

### Retornar o nome das pessoas ligadas a George que tem o nome começando com C em maiúsculo:

```
1 MATCH (george {name: 'George'})<--(otherPerson)
2 WITH otherPerson, toUpper(otherPerson.name) AS upperCaseName
3 WHERE upperCaseName STARTS WITH 'C'
4 RETURN otherPerson.name
```

## MATCH (Pesquisa)

**O nome da pessoa conectada a 'David' com pelo menos um relacionando partindo de si:**

```
1 MATCH (david {name: 'David'})--(otherPerson)-->()
2 WITH otherPerson, count(*) AS foaf
3 WHERE foaf > 1
4 RETURN otherPerson.name
```



## MATCH (Pesquisa)

**Amigos dos meus amigos (eu sendo "Pedro"). Retorno: "João":**

```
1 MATCH (:Pessoa{nome: "Pedro"})<-[:AMIGO]-()-[:AMIGO]-(p) return p
```



## MATCH (Pesquisa)

### SQL:

```
1 SELECT p.ProductName, p.UnitPrice FROM products AS p
2 WHERE p.ProductName = "Chocolate";
```

### *Cypher:*

```
1 MATCH (p:Product)
2 WHERE p.productName = "Chocolate"
3 RETURN p.productName, p.unitPrice;
```

## MATCH (Pesquisa)

### SQL:

```
1 SELECT p.ProductName, p.UnitPrice FROM products AS p
2 WHERE p.ProductName != "Chocolade";
```

### Cypher:

```
1 MATCH (p:Product)
2 WHERE NOT p.productName = "Chocolade"
3 RETURN p.productName, p.unitPrice;
```

Cypher tem ainda os operadores lógicos AND, OR, XOR, e NOT e relacionais >, >=, < e <=

## MATCH (Pesquisa)

```
1 // Retorna todos os users que possuem a propriedade birthdate
2 MATCH (p:Person)
3 WHERE exists(p.birthdate)
4 RETURN p.name;
```

```
1 // Retorna todos os nós que tem o relacionamento all WORKS_FOR com a
   propriedade property
2 MATCH (p:Person)-[rel:WORKS_FOR]->(c:Company)
3 WHERE exists(rel.startYear)
4 RETURN p, rel, c;
```

## MATCH (Pesquisa)

### SQL:

```
1 SELECT p.ProductName, p.UnitPrice
2 FROM products as p
3 WHERE p.ProductName IN ("Chocolade", "Chai");
```

### *Cypher:*

```
1 MATCH (p:Product)
2 WHERE p.productName IN ["Chocolade", "Chai"]
3 RETURN p.productName, p.unitPrice;
```

## MATCH (Pesquisa)

### SQL:

```
1 SELECT p.ProductName, p.UnitPrice FROM products AS p
2 WHERE p.ProductName LIKE "C%" AND p.UnitPrice > 100;
```

### *Cypher:*

```
1 MATCH (p:Product)
2 WHERE p.productName STARTS WITH "C" AND p.unitPrice > 100
3 RETURN p.productName, p.unitPrice;
```

## MATCH (Pesquisa)

```
1 // Verifica se a propriedade contém "a"
2 MATCH (p:Person)
3 WHERE p.name CONTAINS 'a'
4 RETURN p.name;
```

```
1 // Verifica se a propriedade termina com "n"
2 MATCH (p:Person)
3 WHERE p.name ENDS WITH 'n'
4 RETURN p.name;
```

## MATCH (Pesquisa)

É possível ainda usar expressões regulares:

```
$ MATCH (p:Person) WHERE p.name =~ 'Jo.*' RETURN p.name
```

p.name
"John"
"Joe"

Started streaming 2 records after 1 ms and completed after 2 ms.



## MATCH (Pesquisa)

### Join Records, Distinct Results:

#### SQL:

```
1 SELECT DISTINCT c.CompanyName FROM customers AS c
2 JOIN orders AS o ON (c.CustomerID = o.CustomerID)
3 JOIN order_details AS od ON (o.OrderID = od.OrderID)
4 JOIN products AS p ON (od.ProductID = p.ProductID)
5 WHERE p.ProductName = "Chocolade";
```

#### *Cypher:*

```
1 MATCH (p:Product {productName:"Chocolade"})<-[:PRODUCT]-(:Order)
2 <-[:PURCHASED]-(c:Customer) RETURN distinct c.companyName;
```

## MATCH (Pesquisa)

### Aggregation, Grouping:

#### SQL:

```
1 SELECT e.EmployeeID, count(*) AS Count
2 FROM Employee AS e
3 JOIN Order AS o ON (o.EmployeeID = e.EmployeeID)
4 GROUP BY e.EmployeeID
5 ORDER BY Count DESC LIMIT 10;
```

#### *Cypher:*

```
1 MATCH (:Order)<-[:SOLD]-(e:Employee)
2 RETURN e.name, count(*) AS cnt
3 ORDER BY cnt DESC LIMIT 10
```

## MATCH (Atualização)

### Atualizando Propriedade:

```
1 MATCH (p:Pessoa) WHERE ID(p) = 2218 SET p.nome = "Marlon Sousa"
2 RETURN *
```

### Adicionando Propriedade:

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

### Removendo Propriedade:

```
1 // delete property using REMOVE keyword
2 MATCH (n:Person {name: "Igor"}) REMOVE n.profissao
3
4 // delete property with SET to null value
5 MATCH (n:Pessoa) WHERE n.nome = "Igor" SET n.profissao = null
```

## MATCH (Atualização)

### Com datas:

```
1 MATCH (p:Person {name: "Jennifer"})
2 SET p.birthdate = date("1980-01-01")
3 RETURN p
```

### Atualizando propriedades de um relacionamento (aresta):

```
1 MATCH (:Person {name: "Jennifer"})-[rel:WORKS_FOR]-(:Company {name:
   "Neo4j"})
2 SET rel.startYear = date({year: 2018})
3 RETURN rel
```

## DELETE/DETACH DELETE

### Deletando um relacionamento:

```
1 MATCH (j:Person {name: "Jennifer"})-[r:IS_FRIENDS_WITH]->(m:Person
   {name: "Mark"})
2 DELETE r
```

### Deletando um nodo:

```
1 MATCH (m:Person {name: "Mark"}) DELETE m
```

### Deletando nodo e seus relacionamentos:

```
1 MATCH (m:Person {name: "Mark"})
2 DETACH DELETE m
```

## Agenda

1 Neo4j

2 *Cypher*

3 Client Java

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

## Introdução

```
1 public class Example {
2     public static void main(String args[]) {
3         Driver driver = GraphDatabase.driver("bolt://<HOST>:<BOLTPORT>",
4             AuthTokens.basic("<USERNAME>", "<PASSWORD>"));
5         try (Session session =
6             driver.session(SessionConfig.forDatabase("neo4j"))) {
7             String cypherQuery =
8                 "MATCH (p:Product)-[:PART_OF]->(:Category)-[:PARENT*0..]->" +
9                 "(:Category {categoryName:$category})" +
10                "RETURN p.productName as product";
11            var result = session.readTransaction(
12                tx -> tx.run(cypherQuery, parameters("category", "Dairy
13                    Products"))).list();
14            for (Record record : result) {
15                System.out.println(record.get("product").asString());
16            }
17        }
18        driver.close();
19    }
20 }
```

## 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

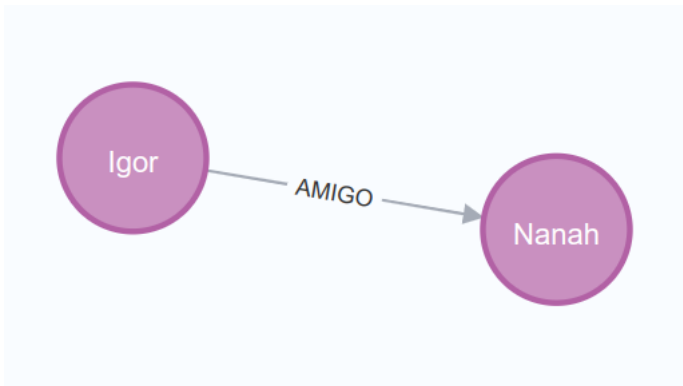
## Criando um relacionamento entre o nodo p1 e p2:

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

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

## CREATE

### Resultado:



## MATCH

## Retornando o nome de uma pessoa dado um cpf:

```
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

## Buscando uma pessoa por cpf:

```
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 - Cypher

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