

# Hands On Neo4j

---

IGOR ROZANI

# Igor Rozani

---



IgorRozani



IgorRozani



@IgorRozani



IgorRozani

# Agenda

---

- Visão geral sobre banco de grafos e Neo4j
- Criação, edição e exclusão de dados com Cypher
- Leitura e manipulação de dados
- Comandos avançados
- Mercado de trabalho

# Visão geral sobre banco de grafos e Neo4j

---

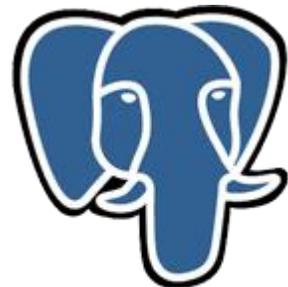
# Tipo de bancos

---

## SQL



ORACLE®



PostgreSQL



## NOSQL



cassandra



# Tipos de NoSQL

**Key Value**



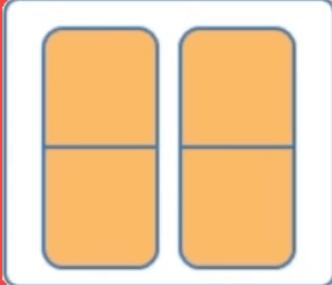
**Example:**  
Riak, Tokyo Cabinet, Redis server, Memcached, Scalaris

**Document-Based**



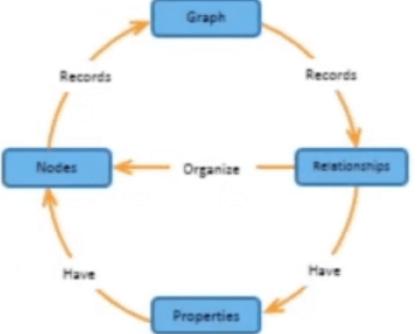
**Example:**  
MongoDB, CouchDB, OrientDB, RavenDB

**Column-Based**



**Example:**  
BigTable, Cassandra, Hbase, Hypertable

**Graph-Based**



**Example:**  
Neo4J, InfoGrid, Infinite Graph, Flock DB

# Bancos de grafos

---



# Bancos de grafos

---



SQL



Cypher

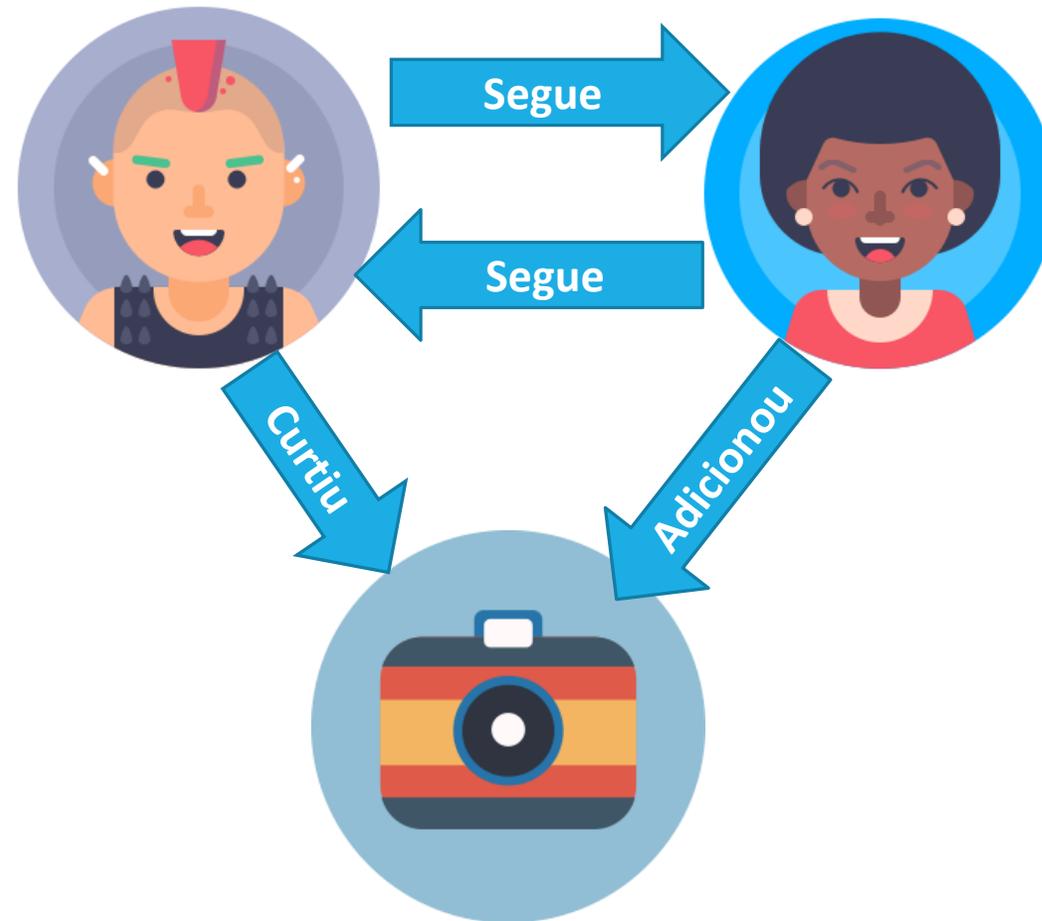
# Características do Neo4j

---

- Schemaless: não possui estrutura e nem tipo de dados definido;
- Index-free: não utiliza índices para organizar e buscar os dados, todos os relacionamentos são gravados com um ponteiro para o próximo nó;
- ACID (Atomic, Consistent, Isolated, Durable): tudo é gravado ou nada;
- Suporte a cluster.

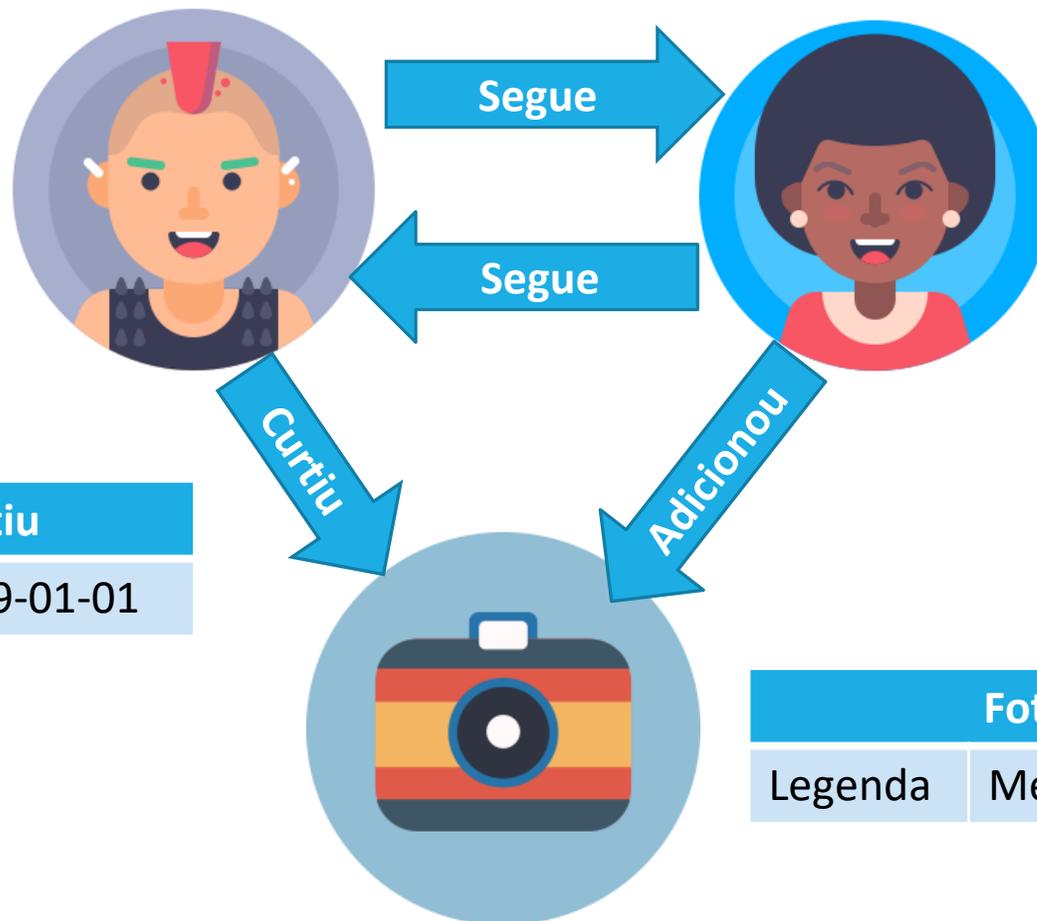
# Estrutura de grafos

---



# Estrutura de grafos

Pessoa	
Nome	Gabriel
País	Canadá
Estado civil	Namorando



Curtiu	
Data	2019-01-01

Pessoa	
Nome	Carol
País	Brasil

Foto	
Legenda	Meu cachorro

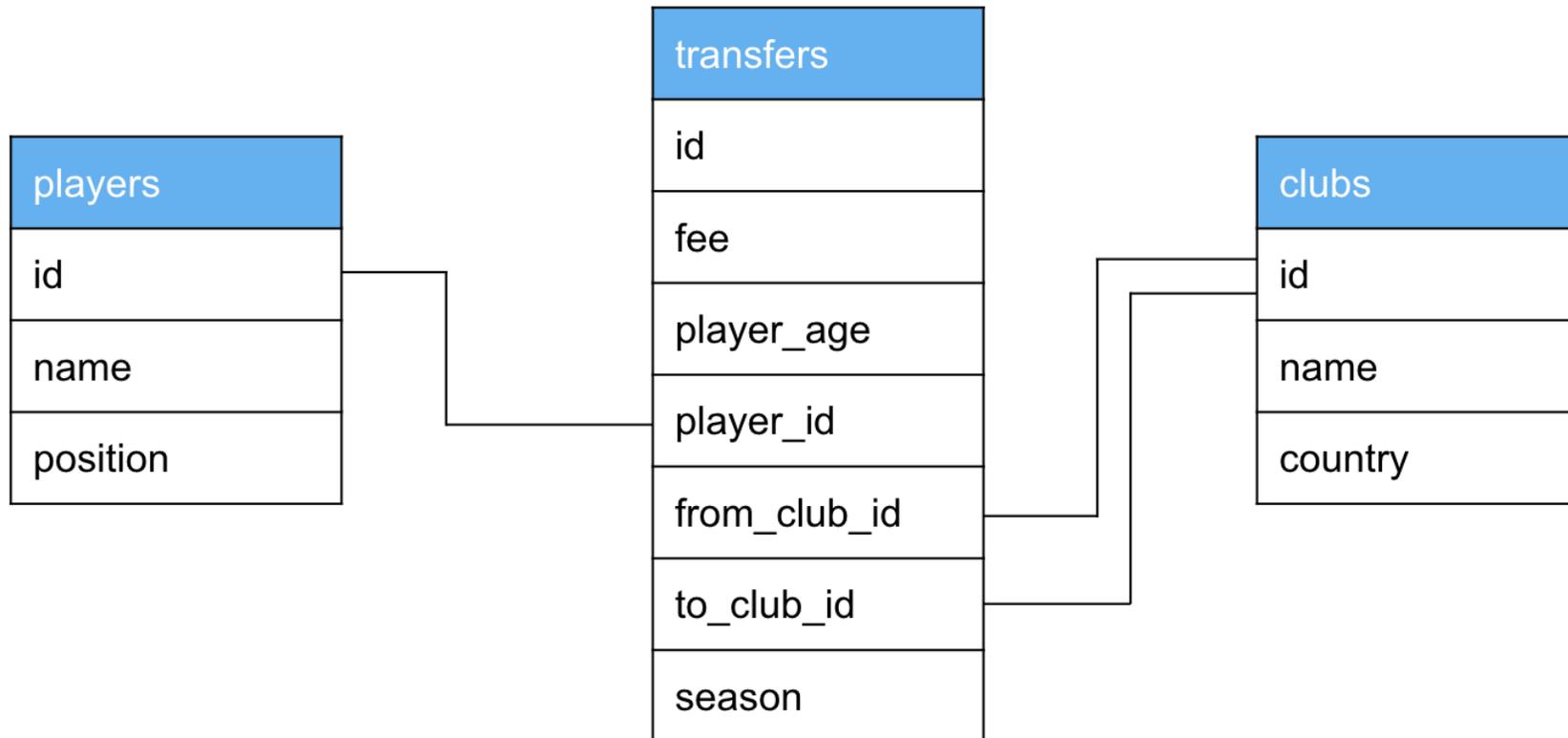
# Estrutura de grafos

---

<b>Relacional</b>	<b>Grafo</b>
Linha	Nós
Joins	Relacionamentos
Nome da tabela	Labels
Coluna	Propriedades

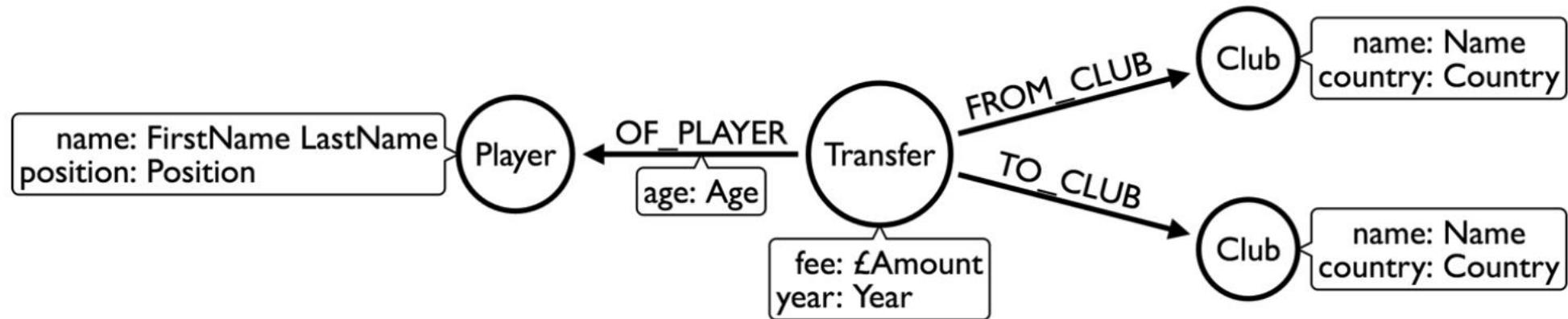
# Estrutura de grafos

---



# Estrutura de grafos

---



# Exercício de modelagem

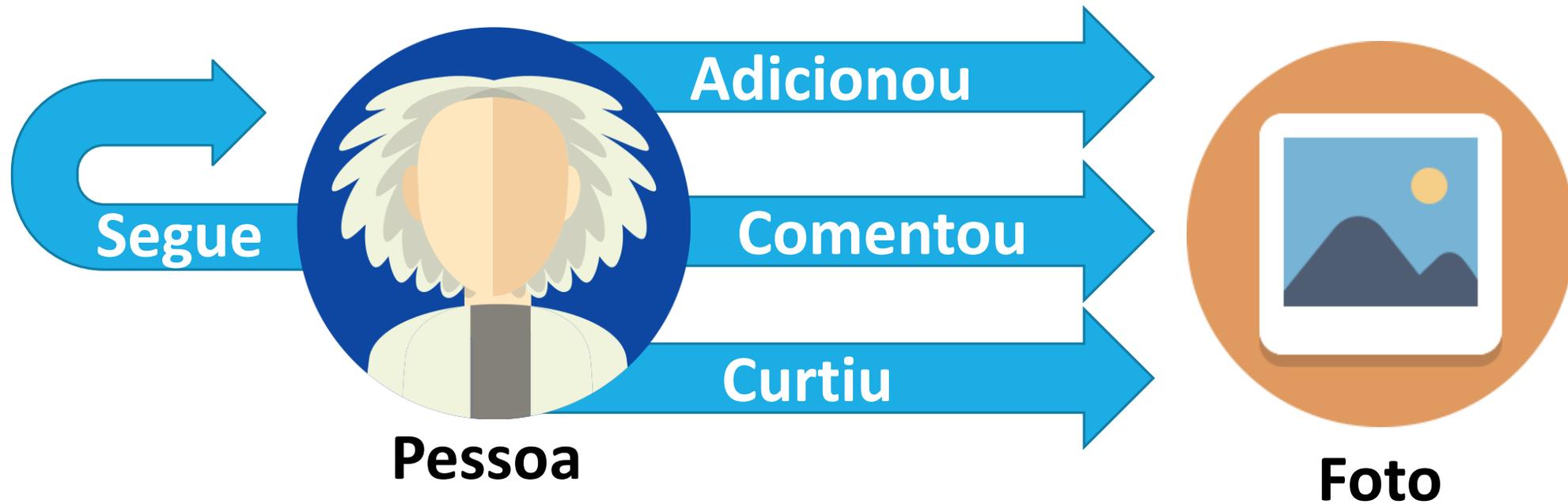
---

A sua empresa foi contratada para criar um sistema similar ao Instagram, que se chamará Fotogram. Foi combinado com o cliente as seguintes funcionalidades para o aplicativo:

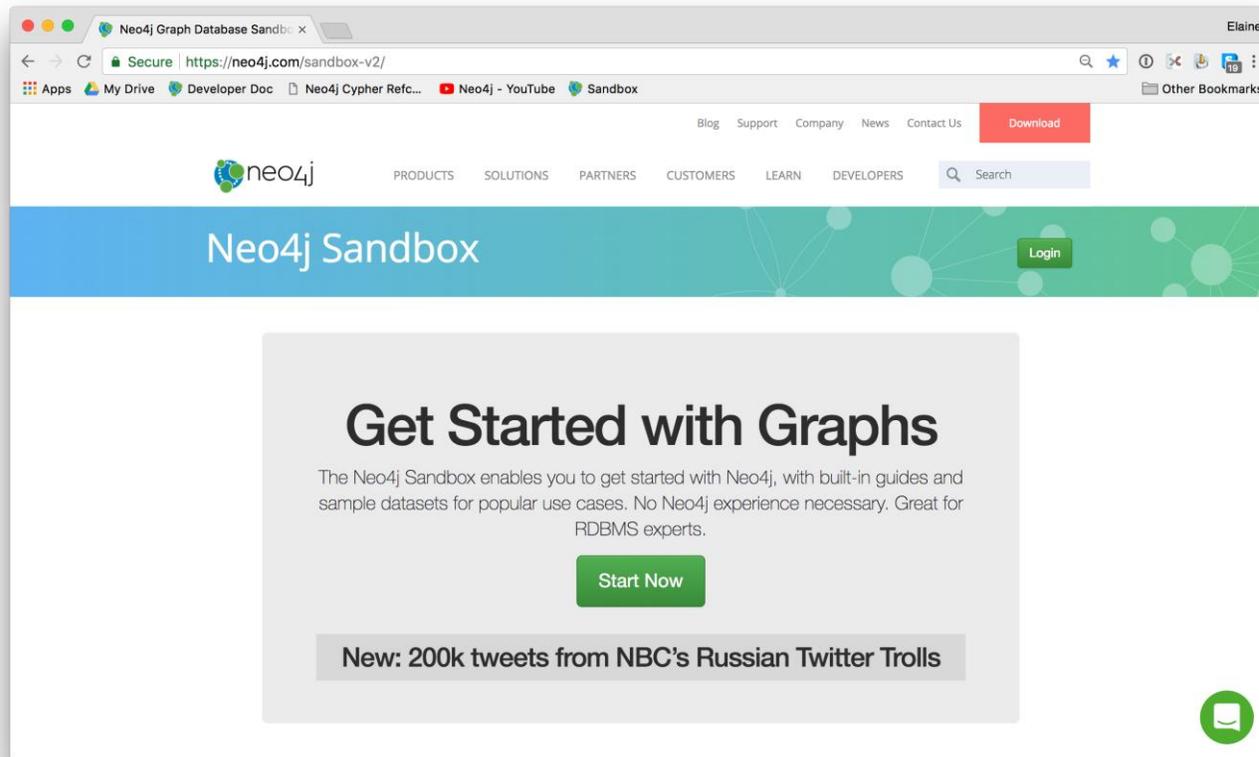
- O usuário terá um perfil na rede social;
- O usuário poderá seguir outros usuários;
- O usuário poderá postar fotos;
- O usuário poderá comentar na foto, os comentários não terão a funcionalidade de responder o comentário;
- O usuário poderá curtir as fotos.

# Estrutura do banco do exercício

---



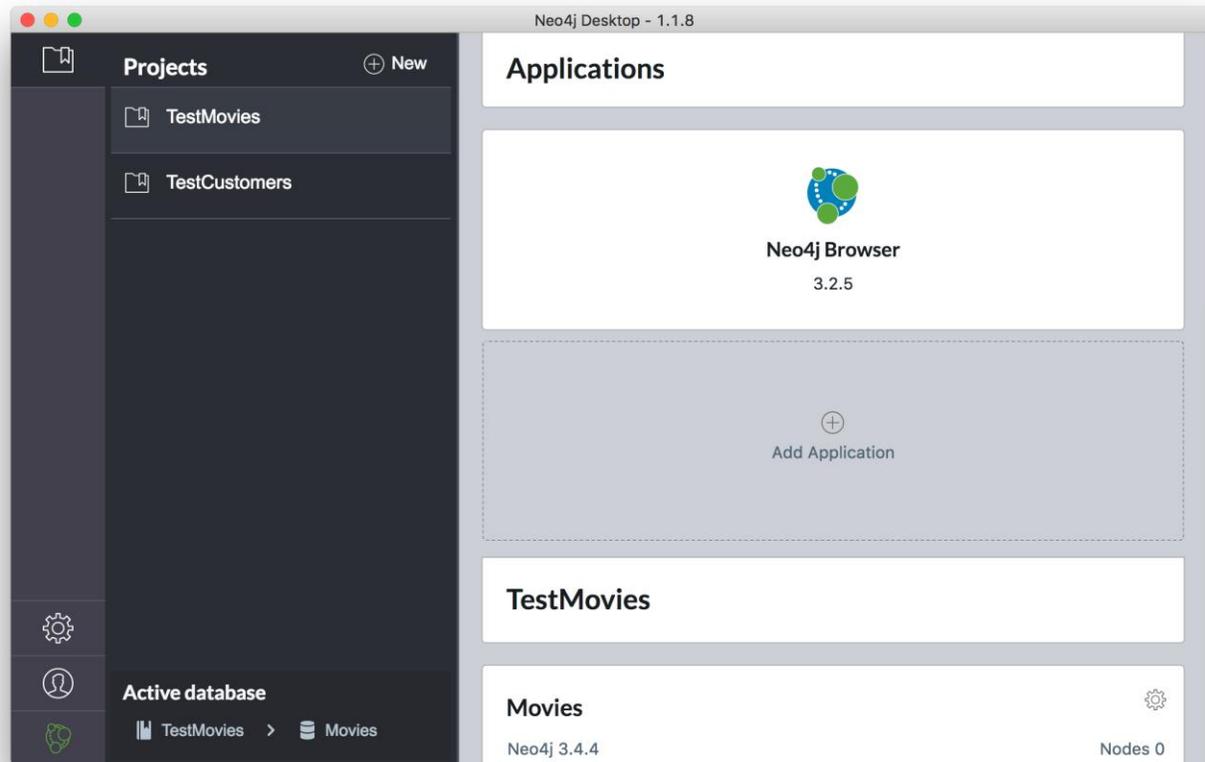
# Neo4j Sandbox



<https://neo4j.com/sandbox-v2/>

# Neo4j Desktop

---



Disponível gratuitamente em  
<https://neo4j.com/download/>

# Demonstração do Neo4j Desktop



# Criação, edição e exclusão de dados com Cypher

---

# Criação de um nó

---

Pessoa	
Nome	Gabriel
País	Canadá
Estado civil	Namorando



```
CREATE (:Pessoa {nome: 'Gabriel', pais: 'Canadá', estadoCivil: 'namorando'})
```

# Criação de um nó

---

CREATE (:Pessoa {nome: 'Gabriel', pais: 'Canadá', estadoCivil: 'namorando'})

Comando  
criação

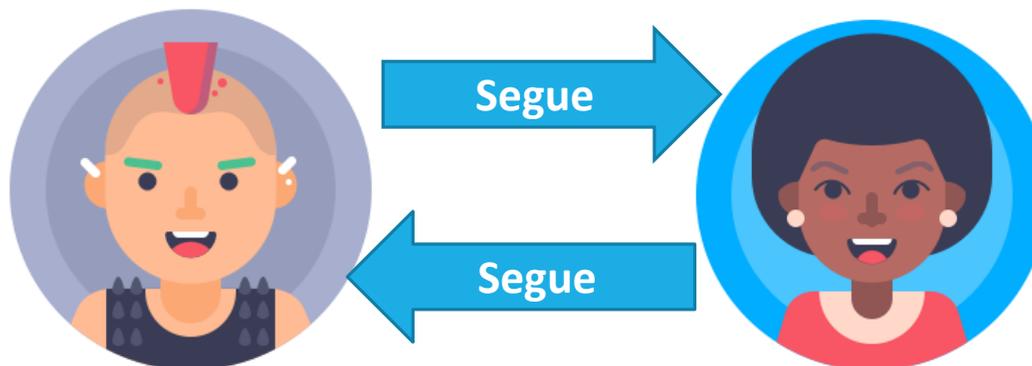
Label

Propriedades

# Criação de um relacionamento

---

Pessoa	
Nome	Gabriel
País	Canadá
Estado civil	Namorando



Pessoa	
Nome	Carol
País	Brasil

```
MATCH (p1:Pessoa{nome: 'Gabriel'}), (p2:Pessoa{nome: 'Carol'})  
CREATE (p1)-[:SEGUE]->(p2)
```

# Criação de um relacionamento

---

Comando busca

Filtro do primeiro nó

Filtro do segundo nó

MATCH (p1:Pessoa{nome: 'Gabriel'}), (p2:Pessoa{nome: 'Carol'})

Indicação de relacionamento

Direção do relacionamento

CREATE (p1)-[:SEGUE]->(p2)

Comando criação

Nó 1

Dados do relacionamento

Nó 2

# Criar nó e relacionamento ao mesmo tempo

---

MATCH (p:Pessoa{usuario:'mrosa'})

CREATE (f:Foto{legenda:'Mais que amigas, friends'}),

(p)-[:ADICIONOU]->(f)

**ATENÇÃO:** Só funciona se ambos os comandos são executados ao mesmo tempo

# Atualizar um registro

---



Pessoa	
Nome	Carol
País	Brasil

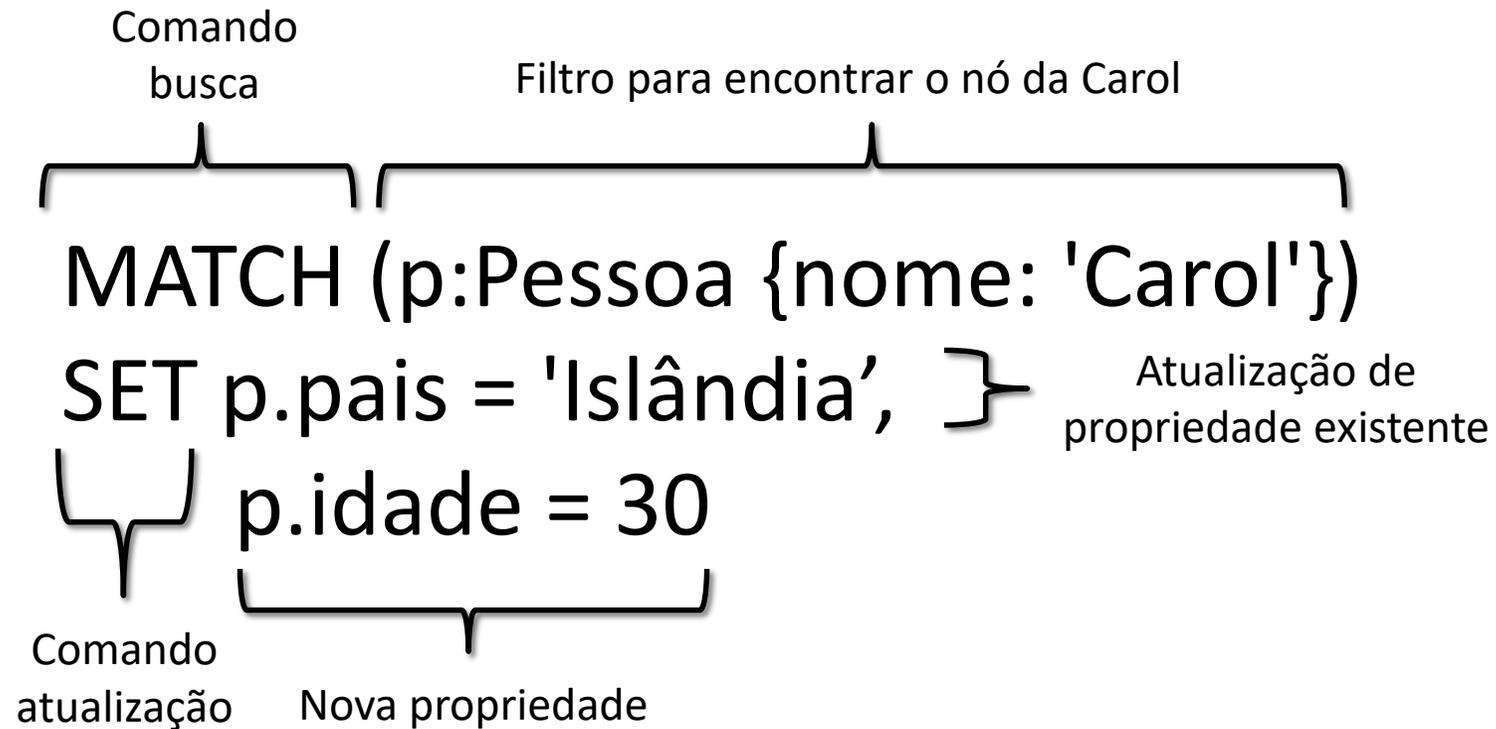


Pessoa	
Nome	Carol
País	Islândia
Idade	30

```
MATCH (p:Pessoa {nome: 'Carol'})  
SET p.idade = 30,  
    p.pais = 'Islândia'
```

# Atualizar um registro

---



# Atualizar um registro

---

MATCH (p:Pessoa{nome : 'Carol '})

SET p = {pais : 'Islândia', nome: 'Carol', idade: 30}

# Adicionar uma propriedade

---

```
MATCH (p:Pessoa {nome: 'Carol'})  
SET p.idade = 30
```

```
MATCH (p:Pessoa {nome: 'Carol'})  
SET p += {idade:30}
```

# Remover uma propriedade

---

MATCH (p:Pessoa{nome : 'Carol '})

SET p = {pais : 'Islândia', nome: 'Carol'}

MATCH (p:Pessoa{nome : 'Carol '})

SET p.idade = null

# Adicionar/Remove um label

---

```
MATCH (p:Pessoa{nome : 'Carol '})
```

```
SET p:Pago
```

```
MATCH (p:Pessoa{nome : 'Carol '})
```

```
REMOVE p:Pago
```

**Adicionar e/ou remover Label funciona apenas para nodes,  
para relacionamentos infelizmente é necessário excluir e criar novamente**

# Excluir um nó

---

Comando busca

Filtro para encontrar o nó da Carol

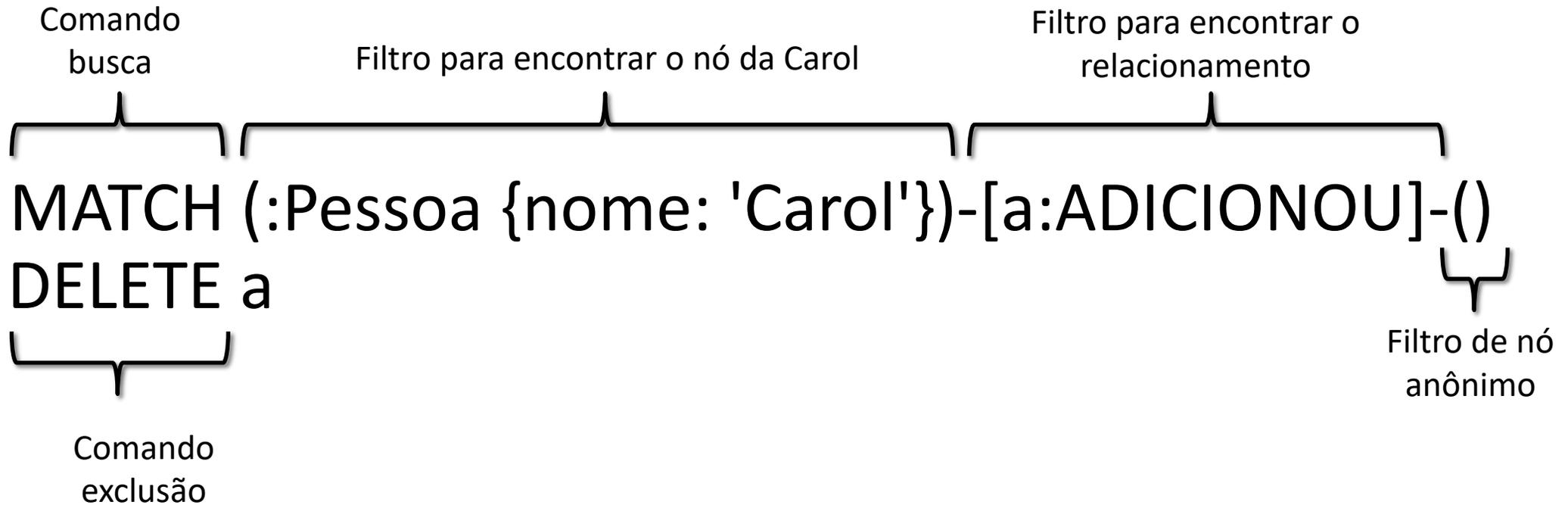
```
MATCH (p:Pessoa {nome: 'Carol'})  
DELETE p
```

Comando exclusão

**Este comando só funciona se o nó não possuir relacionamentos**

# Excluir um relacionamento

---



# Excluir um nó e seus relacionamentos

---

Comando  
busca

Filtro para encontrar o nó da Carol

```
MATCH (p:Pessoa {nome: 'Carol'})  
DETACH DELETE p
```

Comando para exclusão de nó e  
seus relacionamentos

The diagram illustrates the components of the Cypher query. A bracket above the first part of the query, 'MATCH (p:Pessoa {nome: 'Carol'})', is labeled 'Comando busca'. A second bracket above the same part is labeled 'Filtro para encontrar o nó da Carol'. A third bracket below the second part of the query, 'DETACH DELETE p', is labeled 'Comando para exclusão de nó e seus relacionamentos'.

# Revisão

---

## Criação de nó

```
CREATE (:Pessoa {nome: 'Gabriel', pais: 'Canadá', estadoCivil: 'namorando'})
```

## Criação de relacionamento

```
MATCH (p1:Pessoa{nome: 'Gabriel'}),  
(p2:Pessoa{nome: 'Carol'})
```

```
CREATE (p1)-[:SEGUE]->(p2)
```

## Edição

```
MATCH (p:Pessoa {nome: 'Carol'})
```

```
SET p.pais = 'Islândia'
```

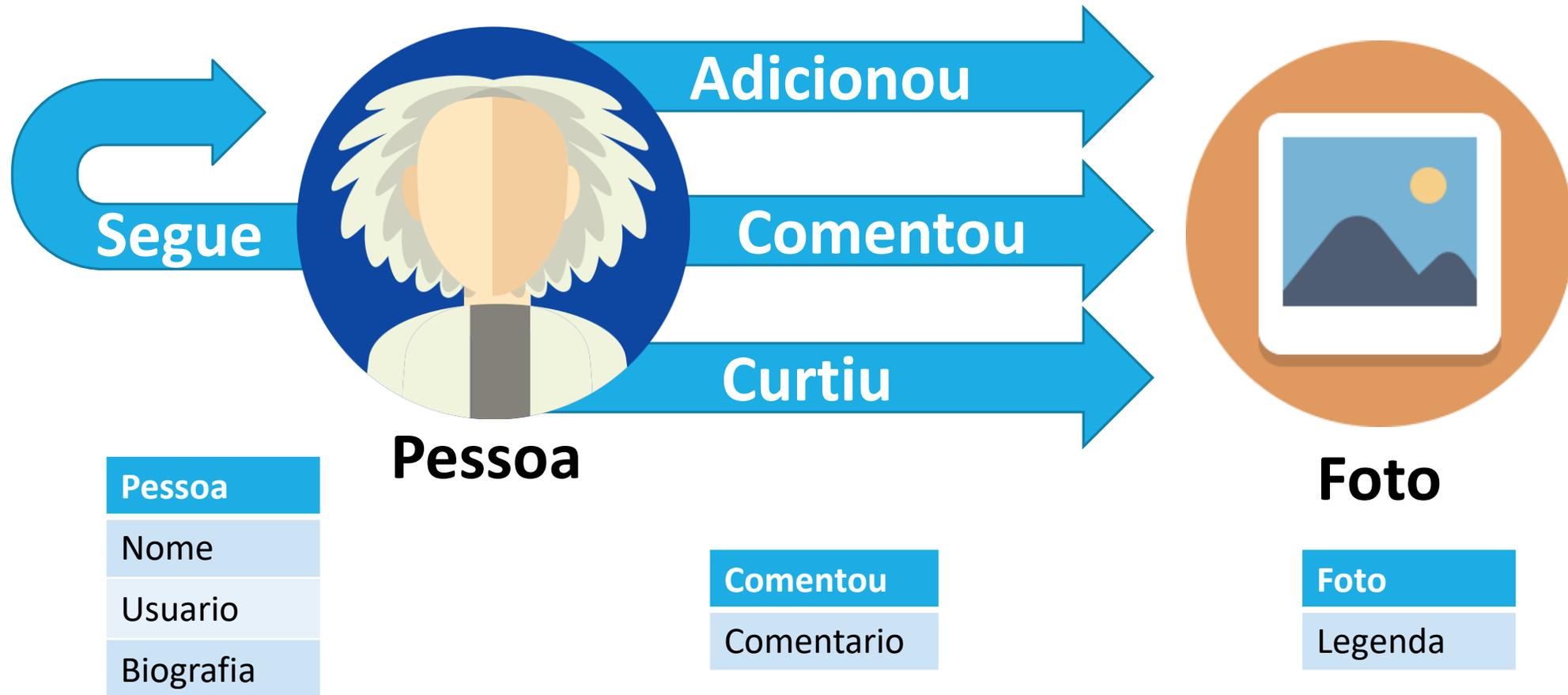
## Exclusão

```
MATCH (p:Pessoa {nome: 'Carol'})
```

```
DETACH DELETE p
```

# Exercício

---



# Exercício

---

Crie o banco de dados do Fotogram, o banco deve possuir os seguintes registros:

- Cadastrar três usuários, sendo dois com usuário, nome e biografia e um com apenas usuário e nome.
- Dois usuários devem se seguir e o terceiro usuário deve seguir apenas um desses usuário.
- Um usuário deve ter três fotos com legenda.
- Todos os usuários devem comentar em uma foto.
- Um usuário, que não tem foto, deve curtir a foto que comentou e uma foto sem comentário.

# Leitura e manipulação de dados

---

# Consultar dados

---

Pessoa	
Nome	Gabriel
País	Canadá
Estado civil	Namorando



```
MATCH (p:Pessoa{nome:'Gabriel'})  
RETURN p
```

# Consultar dados

---

Consulta	Retorno
<pre>MATCH   (p:Pessoa {nome:'Gabriel'}) RETURN p</pre>	<pre>{   "nome": "Gabriel",   "estadoCivil": "Namorando",   "pais": "Canadá" }</pre>
<pre>MATCH   (p:Pessoa {nome:'Gabriel'}) RETURN p.nome</pre>	<pre>"Gabriel"</pre>

# Comando WHERE

---

MATCH (p:Pessoa)

WHERE p.nome = 'Gabriel'

RETURN p

# Comando WHERE

---

```
MATCH (p:Pessoa)
WHERE p.idade >= 18
RETURN p
```

```
MATCH (p:Pessoa)
WHERE 13 <= p.idade <= 18
RETURN p
```

# Comando WHERE

---

MATCH (p:Pessoa)

WHERE p.usuario = 'mrosa' OR p.usuario = 'wsantos'

RETURN p

---

MATCH (p:Pessoa)

WHERE p.usuario IN ['wsantos', 'mrosa']

RETURN p

# Comando EXISTS

---

```
MATCH (f:Foto)
WHERE NOT exists(f.legendada)
RETURN f
```

```
MATCH (f:Foto)
WHERE exists(f.legendada)
RETURN f
```

# Consultar dados

---



Gabriel



Carol

```
MATCH (:Pessoa{nome:"Gabriel"})-[:CURTIU]->(f:Foto)<-[:ADICIONOU]-(:Pessoa{nome:"Carol"})
```

```
RETURN count(f)
```

# Consultar dados

---

Filtro para encontrar as fotos  
que o Gabriel curtiu

Filtro para encontrar as fotos  
que a Carol adicionou

MATCH (:Pessoa{nome:"Gabriel"})-[:CURTIU]->(f:Foto)<-[:ADICIONOU]-(:Pessoa{nome:"Carol"})

RETURN count(f)

Retorno da quantidade  
de fotos curtidas

# Exercício

---

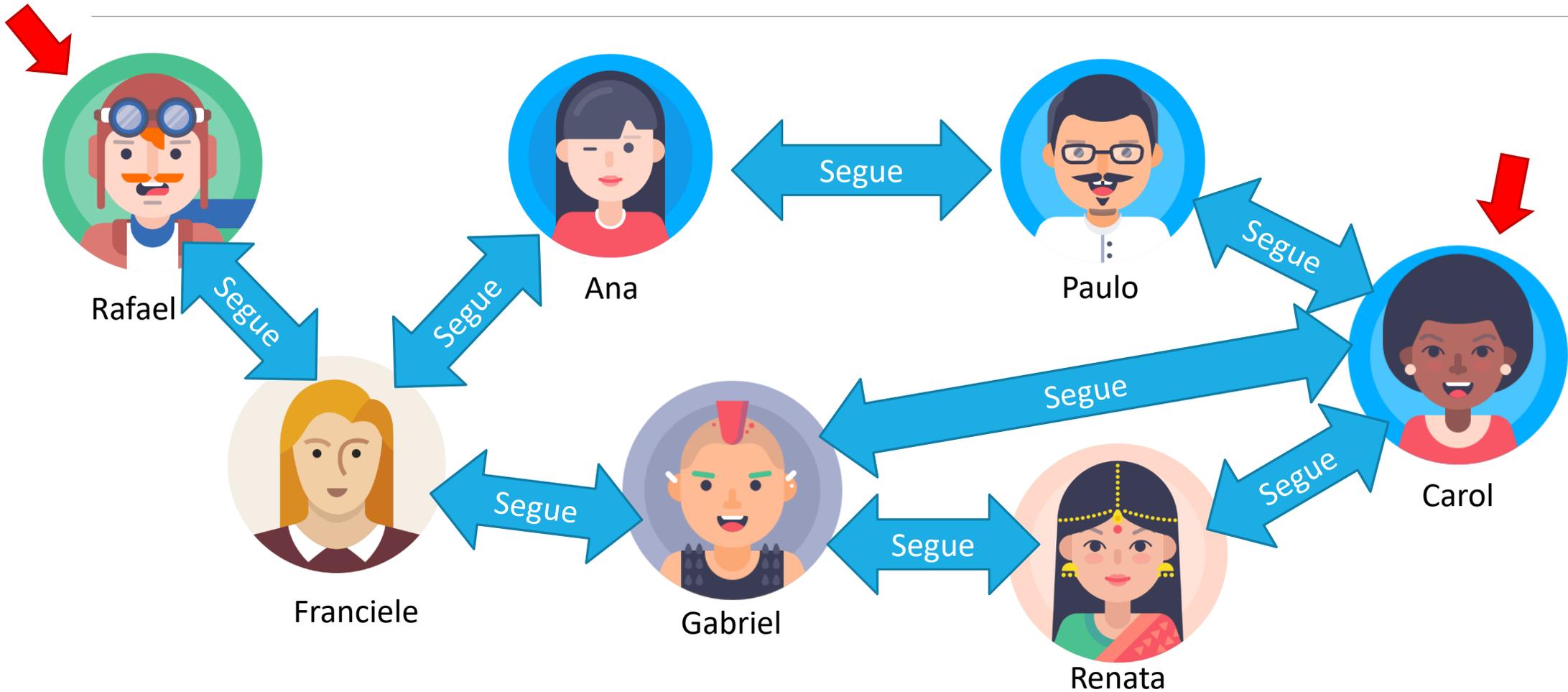
O cliente do Fotogram pediu para exibir algumas informações no perfil dos usuários e você como responsável do banco tem que criar as consultas:

- 1) No perfil é necessário mostrar os dados do usuário, crie uma consulta que traga o nome, usuário e biografia de um usuário.
- 2) Uma consulta que traga todas as fotos de um usuário.
- 3) Exibir a quantidade de pessoas que um usuário segue.
- 4) Exibir a quantidade de pessoas que seguem um usuário.
- 5) Exibir a quantidade de fotos publicadas.
- 6) Usuários que não possuem biografia.

# Comandos avançados

---

# ShortestPath



# ShortestPath

---

MATCH

(p1:Pessoa{nome:'Rafael'}),

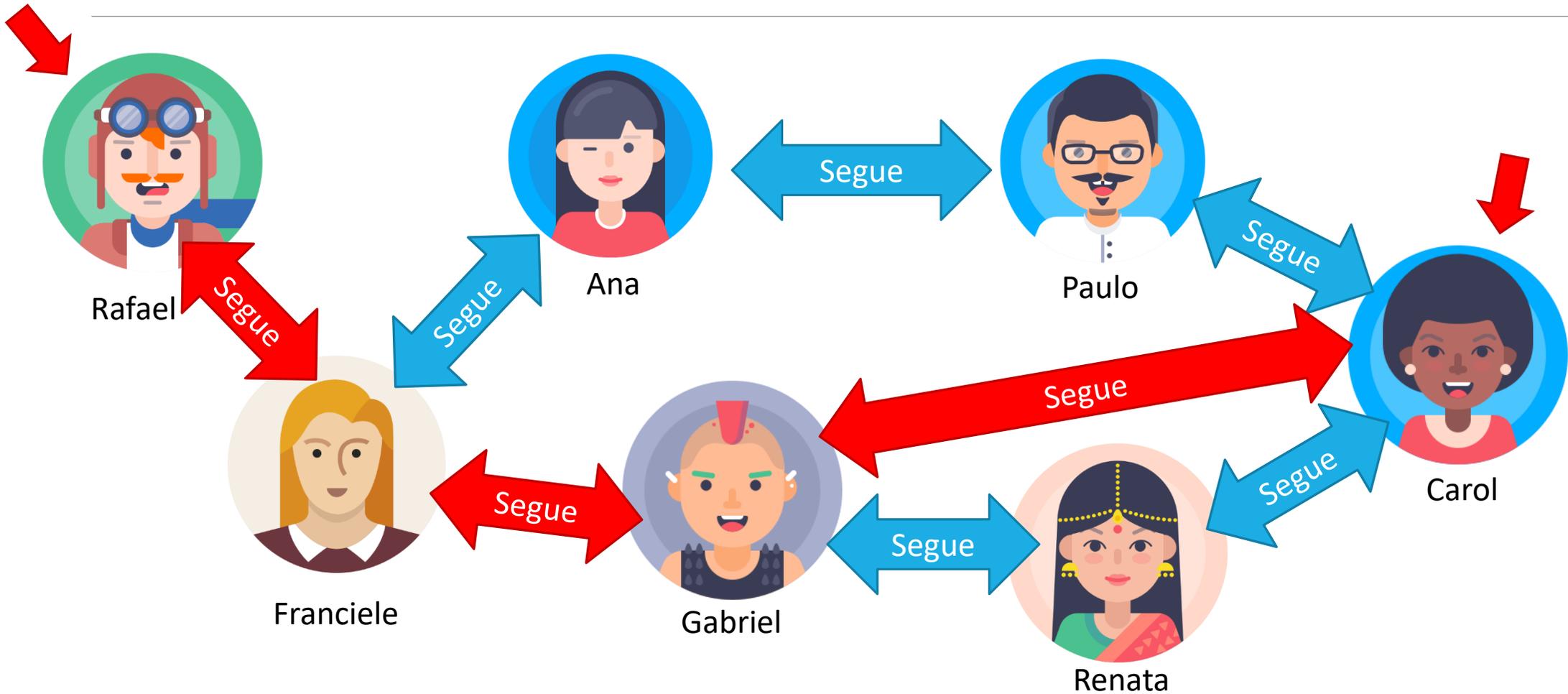
(p2:Pessoa{nome:'Carol'}),

sp = shortestPath((p1)-[:SEGUE\*]-(p2))

RETURN EXTRACT(n in NODES(sp) | n.nome) AS Conexoes

Resultado: ["Rafael", "Franciele", "Gabriel", "Carol"]

# ShortestPath



# Performance

---

Their experiment used a basic social network to find friends-of-friends connections to a depth of five degrees. Their dataset included 1,000,000 people each with approximately 50 friends. The results of their experiment are listed in the table below:

Depth	RDBMS execution time(s)	Neo4j execution time(s)	Records returned
2	0.016	0.01	~2,500
3	30.267	0.168	~110,000
4	1543.505	1.359	~600,000
5	Unfinished	2.132	~800,000

---

**FIGURE 2.2:**

**A performance experiment run between relational databases (RDBMS) and Neo4j shows that graph databases handle data relationships extremely efficiently.**

---

Fonte: Livro Graph Databases for Beginners

# Consultas com relacionamento anônimo

---

Relacionamento anônimo

  
MATCH (p:Pessoa)-->(Pessoa)  
RETURN p

# Comando MERGE

---

Merge é um comando criado para evitar a inserção de dados duplicados, pois ele verifica se o nó ou relacionamento já existe, se existe ele atualiza os dados, se não existe ele cria.

```
MERGE (:Pessoa {usuario: 'mrosa'})
```

O Merge verifica se existe um nó com o Label de Pessoa e com a propriedade usuario com o valor mrosa, se não encontra cria o nó.

# Comando MERGE com ON CREATE

---

O comando ON CREATE com MERGE serve para adicionar propriedades caso o nó não exista e seja criado.

```
MERGE (p:Pessoa {usuario:'ttorres'})
```

```
ON CREATE SET p.nome = 'Tatiane Torres'
```

# Comando MERGE com ON MATCH

---

Já o comando MERGE com ON MATCH é o contrario do ON CREATE, ele serve para atualizar dados extras caso o nó seja encontrado.

```
MERGE (p:Pessoa {usuario:'pferreira'})
```

```
ON MATCH SET p.biografia = 'Eu amo banco de grafos'
```

# Comando MERGE

---

```
MERGE (p:Pessoa {usuario:'ttores'})
```

```
ON CREATE SET p.nome = 'Tatiane Torres'
```

```
ON MATCH SET p.biografia = 'Eu amo banco de grafos'
```

# Exercício de MERGE

---

Use o comando MERGE para atualizar inserir uma biografia para o usuário do Fotogram que não possui.

# Constraints para registros únicos

---

É possível criar constraints no banco para verificar se o nó não está inserindo dados duplicados.

```
CREATE CONSTRAINT ON (pessoa:Pessoa) ASSERT pessoa.usuario IS UNIQUE
```

Para remover uma constraint, basta fazer o mesmo comando com a palavra DROP no lugar de CREATE

```
DROP CONSTRAINT ON (pessoa:Pessoa) ASSERT pessoa.usuario IS UNIQUE
```

# Constraints para campos obrigatórios

---

Outra utilização de constraint é para obrigar nós a terem propriedades específicas.

```
CREATE CONSTRAINT ON (f:Foto) ASSERT exists(f.legenda)
```

Constraints também podem ser aplicadas relacionamentos

```
CREATE CONSTRAINT ON ()-[c:COMENTOU]-() ASSERT  
exists(c.comentario)
```

# Constraint para chave composta

---

Também é possível criar chaves compostas para nós, ao definir a constraint Neo4j não irá permitir inserir registros duplicados para a combinação dos valores da chave composta.

```
CREATE CONSTRAINT ON (n:Pessoa) ASSERT (n.primeiroNome,  
n.ultimoNome) IS NODE KEY
```

# Constraints

---

**Constraint está apenas disponível para bancos que utilizam o Neo4j Enterprise Edition, que é a versão paga.**

# Exercício de constraints

---

Crie uma constraint para evitar o cadastro de usuários com a propriedade usuario duplicado.

# DISTINCT

---

Similar ao SQL, em Cypher é possível retornar valores sem redundância com o comando DISTINCT.

```
MATCH (p:Pessoa)  
RETURN DISTINCT p.nome
```

# Collect

---

Com a função Collect é possível agrupar os valores em uma lista, simplificando o retorno dos dados.

```
MATCH (p:Pessoa)
```

```
RETURN collect(p.nome)
```

# Mercado de trabalho

---

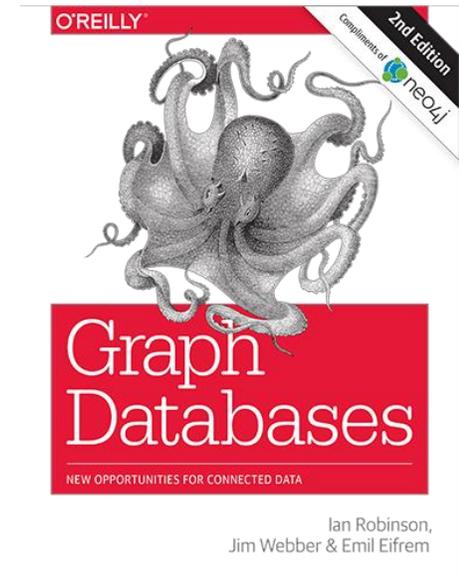
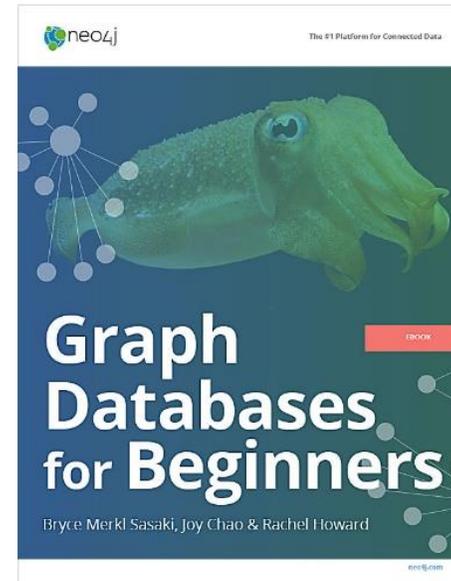
# Como começar?

---

Documentação

Neo4j GraphGists

Online meetups



# Certificação

---

**Preço:** grátis

**Questões:** 80

**Tempo de prova:** 1 hora

**Nota mínima:** 80%

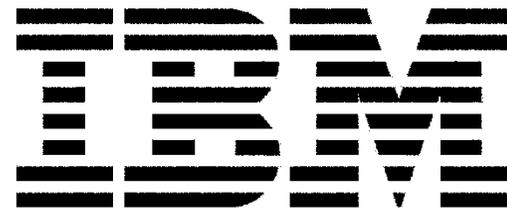
**Formato:** perguntas de múltiplas escolhas

**Limite de tentativas?:** infinito enquanto não passar



# Empresas que utilizam

---



# Vagas

---

	Brasil	Mundo
Neo4j	7	2487
Graph database	10	8438

Pesquisa realizada em vagas abertas no LinkedIn

# Quando usar?

---

Redes sociais

Sistema de recomendação

Sistema de logística

Detecção de fraude

Controle de acesso



“Vou usar banco de grafos pra tudo”

Obrigado!

---