



# Minicurso: Projeto de Bancos de Dados NoSQL (**Prática**)

**Angelo Augusto Frozza**  
**Geomar André Schreiner**  
**Ronaldo dos Santos Mello**

# Agenda



1. Prática MongoDB
2. Prática Neo4J

# Agenda



**1. Prática MongoDB**

2. Prática Neo4J

# MongoDB



- Baseado em estruturas de documento (JSON, BSON)
- Possui uma estrutura hierárquica
- BD mais famoso dentre os NoSQL
  - 5º BD mais utilizado no mundo!

# MongoDB

- Baseado em estruturas de documento (JSON, BSON)
- Possui uma estrutura hierárquica
- BD mais famoso dentre os NoSQL
  - 5º BD mais utilizado no mundo!



# MongoDB

- Baseado em estruturas de documento (JSON, BSON)
- Possui uma estrutura hierárquica
- BD mais famoso dentre os NoSQL
  - 5º BD mais utilizado no mundo!

Oracle	MongoDB
database instance	MongoDB instance
schema	database
table	collection
row	document
rowid	_id
join	DBRef

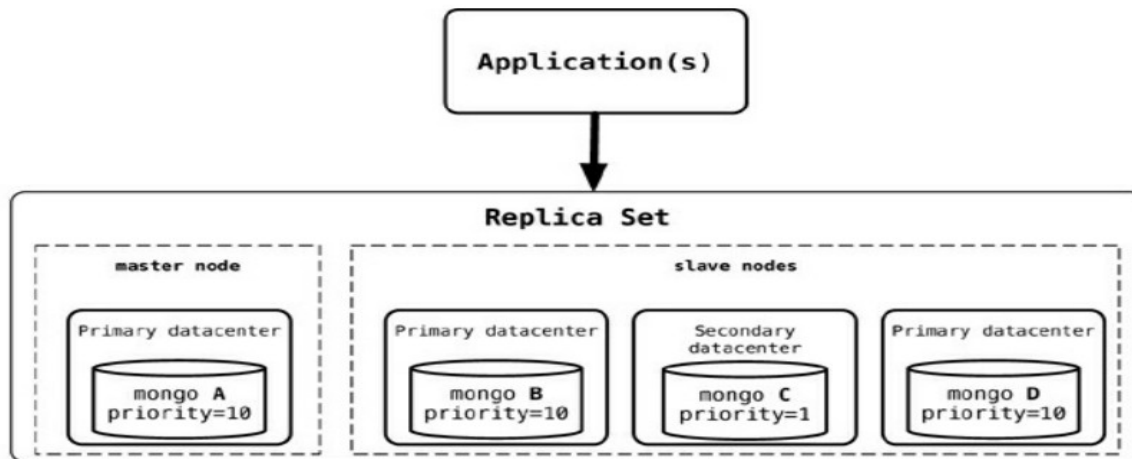
# MongoDB



- Consistência
  - Suporta níveis
    - Apenas escreve no disco
    - Escreve em um nodo e em um número X de réplicas
    - WriteConcern para salvar em todas as replicas antes de retornar
- Transações
  - Por documento

# MongoDB

- Disponibilidade
  - Baseada em replicas
  - Estrutura utilizando um nodo master e outros nodos slave, escritas são tratadas pelo master





# MongoDB

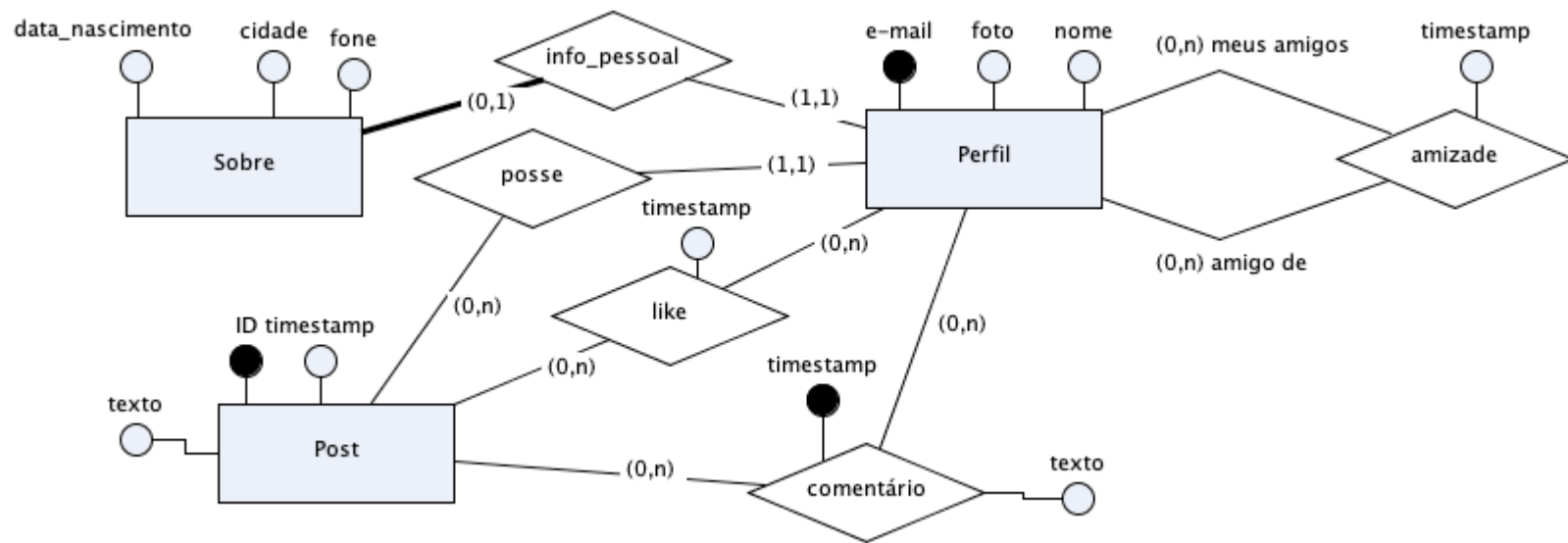


- Instalação
  - Linux
    - Ubuntu
      - `sudo apt install mongodb`
  - Windows
    - [https://fastdl.mongodb.org/windows/mongodb-windows-x86\\_64-4.4.1-signed.msi](https://fastdl.mongodb.org/windows/mongodb-windows-x86_64-4.4.1-signed.msi)
    - Next > Next > Install
- DOCKER IMAGE
  - `docker run -name mongo -d mongo`
  - `docker exec -it mongo bash`
- Online
  - <https://docs.mongodb.com/manual/tutorial/query-documents/>

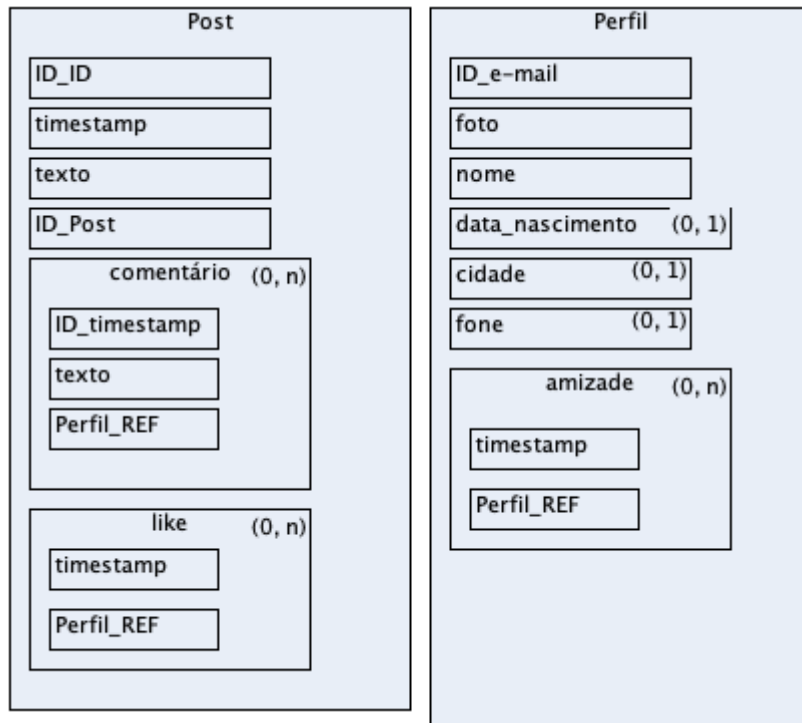
# MongoDB

- Instalação
  - Linux
    - Ubuntu
      - `sudo apt install mongodb`
  - Windows
    - [https://fastdl.mongodb.org/windows/mongodb-windows-x86\\_64-4.4.1-signed.msi](https://fastdl.mongodb.org/windows/mongodb-windows-x86_64-4.4.1-signed.msi)
    - Next > Next > Install
- DOCKER IMAGE
  - `docker run -name mongo -d mongo`
  - `docker exec -it mongo bash`
- Online
  - <https://docs.mongodb.com/manual/tutorial/query-documents/>

# MongoDB - Modelo



# MongoDB - Modelo



# MongoDB



- MongoDB comandos úteis
  - Login
    - **mongo** nome\_base
      - Caso a base não exista será criada
  - Verificar as informações do banco
    - **db.stats()**
  - Mostras as bases do servidor
    - **show dbs**

# MongoDB

- MongoDB Coleções de documentos
  - Buscar
    - **db.getCollection("nome")**
    - Exemplo
      - **db.getCollection("post")**
    - Dica: Usando o shell podemos criar variáveis
      - **var post = db.getCollection("post");**
  - Apagar
    - **db.getCollection('nome').drop()**
    - Exemplo
      - **bd.post.drop()**

# MongoDB

- Inserir

**db.colection.insertOne**({item<sub>1</sub>: valor<sub>1</sub>,... ,item<sub>n</sub>:valor<sub>n</sub>})

**db.colection.insertMany**([ {item<sub>1</sub>: valor<sub>1</sub>, ..., item<sub>n</sub>:valor<sub>n</sub>},  
{item<sub>1</sub>: valor<sub>1</sub>, ..., item<sub>n</sub>:valor<sub>n</sub>}] )

- Exemplo

**db.perfil.insertOne**({nome: "*Pafuncio*",  
cidade: '*Chapecó*'})

- Inserção com \_ID

**db.colection.insertOne**({\_id:<id>, item<sub>1</sub>: valor<sub>1</sub>, ...,  
item<sub>n</sub>:valor<sub>n</sub>})

- Inclusão de data: **new Date** ("yyyy-mm-dd")

# MongoDB

- Inserir

`db.coleção.insert(item_n:valor_n)`  
`db.coleção.insert(valor_1, ..., item_n:valor_n,`  
`{item_1: valor_1, ..., item_n:valor_n}})`

Nome da sua coleção

- Exemplo

`db.perfil.insert({nome: "Pafuncio", cidade:'Chapecó'})`

- Inserção com \_ID

`db.coleção.insertOne({_id:<id>, item_1: valor_1, ...,`  
`item_n:valor_n})`

- Inclusão de data: **new Date** ("yyyy-mm-dd")



# MongoDB

- Inserir

**db.colection.insert**({item<sub>1</sub>: valor<sub>1</sub>,... ,item<sub>n</sub>:valor<sub>n</sub>})

**db.colection.insertMany**([ {item<sub>1</sub>: valor<sub>1</sub>, ..., item<sub>n</sub>:valor<sub>n</sub>},  
{item<sub>1</sub>: valor<sub>1</sub>, ..., item<sub>n</sub>:valor<sub>n</sub>}])

- Exemplo

**db.perfil.insert**({nome: "Pafuncio", c para a coleção '})

- Inserção com \_ID

**db.colection.insertOne**({\_id:<id>, item<sub>1</sub>: valor<sub>1</sub>, ...,  
item<sub>n</sub>:valor<sub>n</sub>})

Deve ser único  
para a coleção

- Inclusão de data: **new Date** ("yyyy-mm-dd")

# MongoDB

- Atualizar

**db.collection.update**(busca, {item1: valor1, ... ,  
itemn:valorn});

- Exemplo

**db.perfil.update**({nome: 'Stephano'}, {nome: 'Cond  
Olaf'});

- CUIDADO!

**db.perfil.update**({nome: 'Stephano'}, {**\$set**: {nome:  
'Cond Olaf'}});

# MongoDB

- Atualizar

**db.collection.update**(busca, {item1: valor1, ... , itemn:valorn});

- Exemplo

**db.perfil.update**({nome: 'Stephano'}, {nome: 'Olaf'});

Sem o set os dados do documento são sobrescritos

- CUIDADO!

**db.perfil.update**({nome: 'Stephano'}, {**\$set**: {nome: 'Cond Olaf'}});

# MongoDB



- Excluir
  - **`db.collection.delete[One|Many](busca)`**
- Exemplo
  - **`db.perfil.deleteOne({nome: 'Cond Olaf'})`**
  - **`db.post.deleteMany({perfil_criador_REF: 2})`**

# MongoDB

- Popular o BD
  - Como instalar daria muito trabalho, vamos utilizar o sistema online :)
- Incluir os dados
  - nome: 'Sunny Baudelaire', data\_nascimento: new Date('2004-02-01'),  
cidade: 'Dream of Stone', amizade: [2,3,8]
  - nome: 'Violet Baudelaire', data\_nascimento: new Date('1994-08-01'),  
cidade: 'Dream of Stone', amizade: [1,3,5,8,7]
- [https://drive.google.com/file/d/1\\_v6Xdp3DSDqFbsjpa-7dzFteCcmDio7C/view?usp=sharing](https://drive.google.com/file/d/1_v6Xdp3DSDqFbsjpa-7dzFteCcmDio7C/view?usp=sharing)

# MongoDB

- Consultas
  - **db.collection.find(<busca>, projeção);**
  - <busca>
    - 'Documento' com os filtros necessários
    - Operadores
      - > **\$gt** e < **\$lt**
      - >= **\$gte** e <= **\$lte**
  - Projeção
    - Atributos que devem aparecer são sinalizados com 1
- Exemplo
  - db.perfil.find({nome: 'Sunny Baudelaire'}, {\_id: 0, nome: 1, data\_nascimento: 1});**

# MongoDB

- Consultas
  - **db.collection.find(<busca>, projeção);**
  - <busca>
    - 'Documento' com os filtros necessários
    - Operadores
      - > **\$gt** e < **\$lt**
      - >= **\$gte** e <= **\$lte**
  - Projeção
    - Atributos que devem aparecer são sinalizados com 1
- Exemplo
  - db.perfil.find({nome: 'Sunny Baudelaire'}, {\_id: 0, nome: 1, data\_nascimento: 1});**

# MongoDB

- Consultas

- db.collection.find**(busca, projeção),

- <busca>

- Operadores

- **\$and** : [{operador}, {operador}]

- **\$or** : [{operador}, {operador}]

- Exemplo

- db.perfil.find**({**\$or**: [{nome: 'Klaus Baudelaire'},  
{data\_nascimento: **{ \$gt: new Date('1999-11-3')}**}]}, {nome: 1,  
amizade: 1});



# MongoDB

- Consultas
  - **db.collection.find**(busca, projeção),
  - <busca>
    - Operadores de vetor (vet: ['a', 'b', 'c'])
      - **db.collection.find**({vet: {\$in: ['a']}}, {});
  - Exemplo
    - db.perfil.find**({amizade: {\$in: ['2']}}, {\_id: 0, nome: 1});

# MongoDB



- Exercícios
  - Apresentar o texto de todos posts e seus respectivos comentários
  - Listar os amigos da “Violet”
  - Apresentar o nome de todos os perfis nascidos antes de 01/02/2000
    - Utilize: `new Date('2000-02-01')`

# Agenda



1. Prática MongoDB

**2. Prática Neo4J**

# Neo4J

The logo for Neo4J, featuring the text "Neo4J" in a bold, black, sans-serif font. Below the text is a horizontal line that is black on the left and orange on the right.

- Possuem seu armazenamento baseado em grafos
  - É otimizado para acesso de relacionamentos e não busca em valores das chaves

## Neo4J

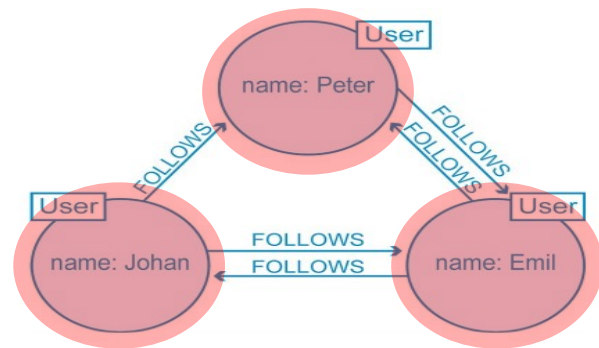
- Possuem seu armazém de dados
- É otimizado para consultas em valores das



o busca

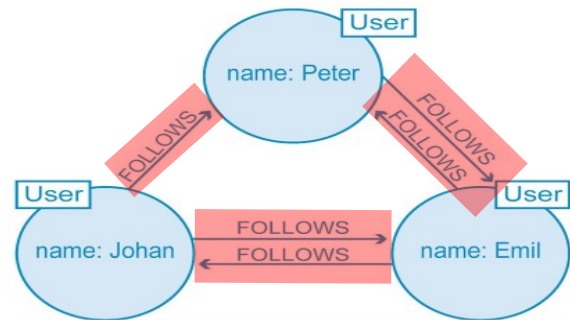
# Neo4J

- Possuem seu armazenamento baseado em grafos
  - É otimizado para acesso de relacionamentos e não busca em valores das chaves
- De maneira geral os BDs deste modelo seguem uma definição clássica de grafos
  - **Nodos:** Objetos que são armazenados



# Neo4J

- Possuem seu armazenamento baseado em grafos
  - É otimizado para acesso de relacionamentos e não busca em valores das chaves
- De maneira geral os BDs deste modelo seguem uma definição clássica de grafos
  - **Nodos:** Objetos que são armazenados
  - **Arestas:** Relacionamentos entre Objetos



# Neo4J

- Representante mais famoso dos BDs Orientados a Grafo
- Linguagem de consulta Cypher
  - Linguagem declarativa
  - Permite buscas por arestas/nodos indexando propriedades
- É multiplataforma
  - Linux, MAC e Windows





# Neo4J

- Representante mais famoso dos BDs Orientados a Grafo
- Linguagem de consulta **Cypher**
  - Linguagem declarativa
  - Permite buscas por arestas/nodos indexando propriedades
- É multiplataforma
  - Linux, MAC e Windows



# Neo4J

- Representante
- Linguagem de
  - Linguagem d
  - Permite busc
- É multiplatafor
  - Linux, MAC e



fo

dades

o4j

# Neo4J

Clear DB Help Share Toggle Viz Options

## Graph Setup:

```
create (Neo:Crew {name:'Neo'}), (Morpheus:Crew {name: 'Morpheus'}), (Trinity:Crew {name: 'Trinity'}), (Cypher:Crew:Matrix {name: 'Cypher'}), (Smith:Matrix {name: 'Agent Smith'}),
(Architect:Matrix {name:'The Architect'}),
(Neo)-[:KNOWS]->(Morpheus), (Neo)-[:LOVES]->(Trinity), (Morpheus)-[:KNOWS]->(Trinity),
(Morpheus)-[:KNOWS]->(Cypher), (Cypher)-[:KNOWS]->(Smith), (Smith)-[:CODED_BY]->(Architect)
```

Query:  
`match (n:Crew)-[r:KNOWS*]-(m) where n.name='Neo' return n as Neo,r,m`

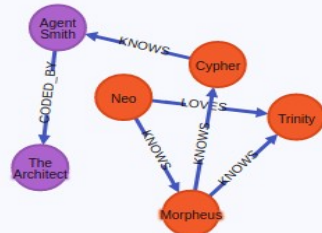
Neo	r	m
(0:Crew {name:"Neo"})	[(0)-[:KNOWS]->(1)]	(1:Crew {name:"Morpheus"})
(0:Crew {name:"Neo"})	[(0)-[:KNOWS]->(1), (1)-[:KNOWS]->(2)]	(2:Crew {name:"Trinity"})
(0:Crew {name:"Neo"})	[(0)-[:KNOWS]->(1), (1)-[:KNOWS]->(3)]	(3:Crew:Matrix {name:"Cypher"})
(0:Crew {name:"Neo"})	[(0)-[:KNOWS]->(1), (1)-[:KNOWS]->(3), (3)-[:KNOWS]->(4)]	(4:Matrix {name:"Agent Smith"})

Query took 122 ms and returned 4 rows. [Result Details](#)

You can modify and query this graph by entering statements in the input field at the bottom.

For some syntax help hit the [Help](#) button.

If you want to share your graph, just do it with [Share](#)



```
match (n:Crew)-[r:KNOWS*]-(m) where n.name='Neo' return n as Neo,r,m
```



# Neo4J

Clear DB Help Share Toggle Viz Options

## Graph Setup:

```
create (Neo:Crew {name:'Neo'}), (Morpheus:Crew {name: 'Morpheus'}), (Trinity:Crew {name: 'Trinity'}), (Cypher:Crew:Matrix {name: 'Cypher'}), (Smith:Matrix {name: 'Agent Smith'}),
(Architect:Matrix {name:'The Architect'}),
(Neo)-[:KNOWS]->(Morpheus), (Neo)-[:LOVES]->(Trinity), (Morpheus)-[:KNOWS]->(Trinity),
(Morpheus)-[:KNOWS]->(Cypher), (Cypher)-[:KNOWS]->(Smith), (Smith)-[:CODED_BY]->(Architect)
```

Query:  
`match (n:Crew)-[:KNOWS*]-(m) where n.name='Neo' return n as Neo,r,m`

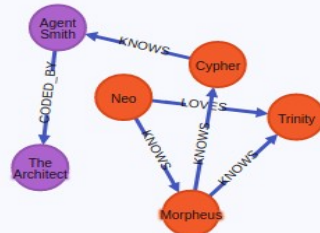
Neo	r	m
(0:Crew {name:"Neo"})	[(0)-[:KNOWS]->(1)]	(1:Crew {name:"Morpheus"})
(0:Crew {name:"Neo"})	[(0)-[:KNOWS]->(1), (1)-[:KNOWS]->(2)]	(2:Crew {name:"Trinity"})
(0:Crew {name:"Neo"})	[(0)-[:KNOWS]->(1), (1)-[:KNOWS]->(3)]	(3:Crew:Matrix {name:"Cypher"})
(0:Crew {name:"Neo"})	[(0)-[:KNOWS]->(1), (1)-[:KNOWS]->(3), (3)-[:KNOWS]->(4)]	(4:Matrix {name:"Agent Smith"})

Query took 122 ms and returned 4 rows. [Result Details](#)

You can modify and query this graph by entering statements in the input field at the bottom.

For some syntax help hit the [Help](#) button.

If you want to share your graph, just do it with [Share](#)



```
match (n:Crew)-[:KNOWS*]-(m) where n.name='Neo' return n as Neo,r,m
```



# Neo4J

Clear DB Help Share Toggle Viz Options

## Graph Setup:

```
create (Neo:Crew {name:'Neo'}), (Morpheus:Crew {name: 'Morpheus'}), (Trinity:Crew {name: 'Trinity'}), (Cypher:Crew:Matrix {name: 'Cypher'}), (Smith:Matrix {name: 'Agent Smith'}),
(Architect:Matrix {name:'The Architect'}),
(Neo)-[:KNOWS]->(Morpheus), (Neo)-[:LOVES]->(Trinity), (Morpheus)-[:KNOWS]->(Trinity),
(Morpheus)-[:KNOWS]->(Cypher), (Cypher)-[:KNOWS]->(Smith), (Smith)-[:CODED_BY]->(Architect)
```

## Query:

```
match (n:Crew)-[r:KNOWS*]-(m) where n.name='Neo' return n as Neo,r,m
```

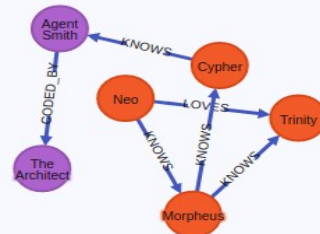
Neo	r	m
{0:Crew {name:"Neo"}} [({0}-[:KNOWS]->{1})]		{1:Crew {name:"Morpheus"}}
{0:Crew {name:"Neo"}} [({0}-[:KNOWS]->{1}), ({1}-[:KNOWS]->{2})]		{2:Crew {name:"Trinity"}}
{0:Crew {name:"Neo"}} [({0}-[:KNOWS]->{1}), ({1}-[:KNOWS]->{3})]		{3:Crew:Matrix {name:"Cypher"}}
{0:Crew {name:"Neo"}} [({0}-[:KNOWS]->{1}), ({1}-[:KNOWS]->{3}), ({3}-[:KNOWS]->{4})]		{4:Matrix {name:"Agent Smith"}}

Query took 122 ms and returned 4 rows. [Result Details](#)

You can modify and query this graph by entering statements in the input field at the bottom.

For some syntax help hit the [Help](#) button.

If you want to share your graph, just do it with [Share](#)



```
match (n:Crew)-[r:KNOWS*]-(m) where n.name='Neo' return n as Neo,r,m
```



# Neo4J

Clear DB Help Share Toggle Viz Options

## Graph Setup:

```
create (Neo:Crew {name:'Neo'}), (Morpheus:Crew {name: 'Morpheus'}), (Trinity:Crew {name: 'Trinity'}), (Cypher:Crew:Matrix {name: 'Cypher'}), (Smith:Matrix {name: 'Agent Smith'}),
(Architect:Matrix {name:'The Architect'}),
(Neo)-[:KNOWS]->(Morpheus), (Neo)-[:LOVES]->(Trinity), (Morpheus)-[:KNOWS]->(Trinity),
(Morpheus)-[:KNOWS]->(Cypher), (Cypher)-[:KNOWS]->(Smith), (Smith)-[:CODED_BY]->(Architect)
```

Query:  
`match (n:Crew)-[r:KNOWS*]-(m) where n.name='Neo' return n as Neo,r,m`

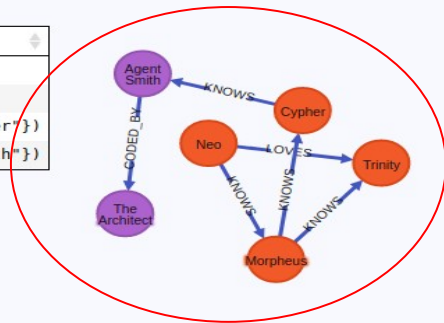
Neo	r	m
(0:Crew {name:"Neo"})	[(0)-[:KNOWS]->(1)]	(1:Crew {name:"Morpheus"})
(0:Crew {name:"Neo"})	[(0)-[:KNOWS]->(1), (1)-[:KNOWS]->(2)]	(2:Crew {name:"Trinity"})
(0:Crew {name:"Neo"})	[(0)-[:KNOWS]->(1), (1)-[:KNOWS]->(3)]	(3:Crew:Matrix {name:"Cypher"})
(0:Crew {name:"Neo"})	[(0)-[:KNOWS]->(1), (1)-[:KNOWS]->(3), (3)-[:KNOWS]->(4)]	(4:Matrix {name:"Agent Smith"})

Query took 122 ms and returned 4 rows. [Result Details](#)

You can modify and query this graph by entering statements in the input field at the bottom.

For some syntax help hit the [Help](#) button.

If you want to share your graph, just do it with [Share](#)



```
match (n:Crew)-[r:KNOWS*]-(m) where n.name='Neo' return n as Neo,r,m
```



# Neo4J

Clear DB Help Share Toggle Viz Options

## Graph Setup:

```
create (Neo:Crew {name:'Neo'}), (Morpheus:Crew {name: 'Morpheus'}), (Trinity:Crew {name: 'Trinity'}), (Cypher:Crew:Matrix {name: 'Cypher'}), (Smith:Matrix {name: 'Agent Smith'}),
(Architect:Matrix {name:'The Architect'}),
(Neo)-[:KNOWS]->(Morpheus), (Neo)-[:LOVES]->(Trinity), (Morpheus)-[:KNOWS]->(Trinity),
(Morpheus)-[:KNOWS]->(Cypher), (Cypher)-[:KNOWS]->(Smith), (Smith)-[:CODED_BY]->(Architect)
```

Query:  
`match (n:Crew)-[:KNOWS*]-(m) where n.name='Neo' return n as Neo,r,m`

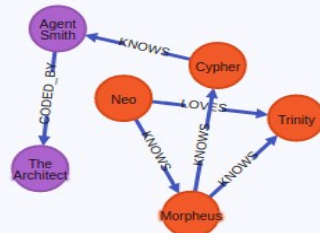
Neo	r	m
{0:Crew {name:"Neo"}} [({0}-[:KNOWS]->({1})]		{1:Crew {name:"Morpheus"}}
{0:Crew {name:"Neo"}} [({0}-[:KNOWS]->({1}), ({1})-[:KNOWS]->({2})]		{2:Crew {name:"Trinity"}}
{0:Crew {name:"Neo"}} [({0}-[:KNOWS]->({1}), ({1})-[:KNOWS]->({3})]		{3:Crew:Matrix {name:"Cypher"}}
{0:Crew {name:"Neo"}} [({0}-[:KNOWS]->({1}), ({1})-[:KNOWS]->({3}), ({3})-[:KNOWS]->({4})]		{4:Matrix {name:"Agent Smith"}}

Query took 122 ms and returned 4 rows. [Result Details](#)

You can modify and query this graph by entering statements in the input field at the bottom.

For some syntax help hit the [Help](#) button.

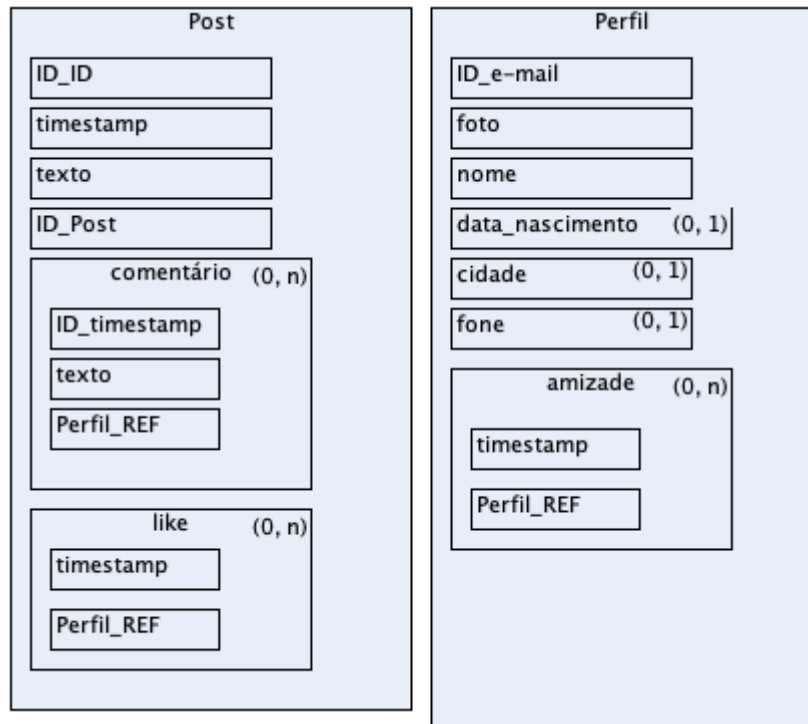
If you want to share your graph, just do it with [Share](#)



```
match (n:Crew)-[:KNOWS*]-(m) where n.name='Neo' return n as Neo,r,m
```



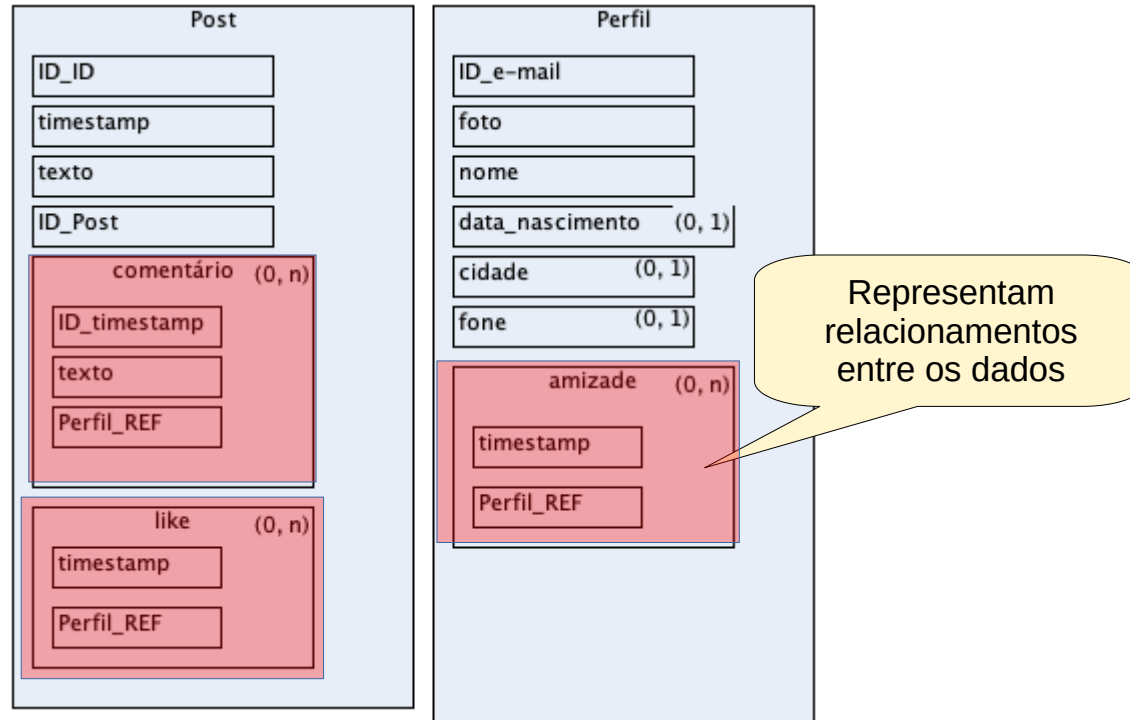
# Neo4J - Modelo



Cada collection vai virar um conjunto de vértices



# Neo4J - Modelo



# Neo4J

- Clique em Clear DB;
- Criar nodos e relações  
**CREATE objeto<sub>1</sub>, objeto<sub>n</sub>**
- Transação ACID!
- Cria um nodo  
**CREATE (s);**



# Neo4J

- Clique em Clear DB;
- Criar nodos e relações  
**CREATE objeto<sub>1</sub>, objeto<sub>n</sub>**
- Transação ACID!
- Cria um nodo  
CREATE (s);  
CREATE (s:Perfil);




# Neo4J

- Clique em Clear DB;
- Criar nodos e relações

**CREATE objeto<sub>1</sub>, objeto<sub>n</sub>**

- Transação ACID!
- Cria um nodo  
CREATE (s);  
CREATE (s:Perfil);  
CREATE (s:Perfil{name: "Sunny"});



Sunny

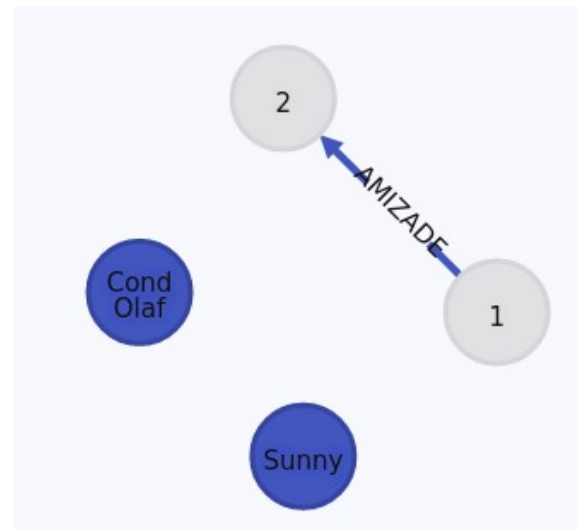
# Neo4J

- Adicionar os nodos 'Sunny' e 'Cond Olaf'  
    (s:Crew {name: "Sunny"})  
    (o:Crew {name: "Cond Olaf"})
- Como adicionar arestas?  
    **CREATE** (nodo1)-[]->(nodo2)  
    **CREATE** (nodo1)-[:Label {atts: "val"}]->(nodo2)  
    **CREATE** (nodo1)-[:Label]->(nodo2)
- Vamos adicionar uma aresta que identifique que Olaf é 'amigo' da Sunny  
    **CREATE** (s)-[:AMIZADE]->(o)

# Neo4J

- Vamos adicionar uma aresta que identifique que Olaf é 'amigo' da Sunny

CREATE (s)-[:AMIZADE]->(o)



# Neo4J

- Vamos adicionar uma aresta que identifique que Olaf é 'amigo' da Sunny
  - Criar o relacionamento junto com os nodos

```
CREATE (s:Perfil{name: "Sunny"}),  
      (o:Perfil{name: "Cond Olaf"}),  
      (o)-[:AMIZADE]->(s)
```

# Neo4J

- Vamos adicionar uma aresta que identifique que Olaf é 'amigo' da Sunny
  - Criar o relacionamento junto com os nodos

```
CREATE (s:Perfil{name: "Sunny"}),  
      (o:Perfil{name: "Cond Olaf"}),  
      (o)-[:AMIZADE]->(s)
```





# Neo4J

- Vamos adicionar uma aresta que identifique que Olaf é 'amigo' da Sunny
  - Selecionar os nodos, e então criar o elemento

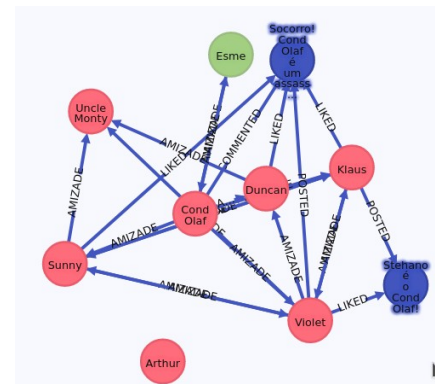
```
MATCH (s:Perfil), (o:Perfil)
  WHERE s.name = "Sunny" and o.name = "Cond Olaf"
CREATE (o)-[:AMIZADE]->(s)
```



# Neo4J

- Popular o BD

```
CREATE (s:Perfil {name: "Sunny"}), (v:Perfil {name: "Violet"}), (k:Perfil {name: "Klaus"}), (o:Perfil {name: "Cond Olaf"}),
(d:Perfil {name: "Duncan"}), (e:Perfil {name: "Esme"}), (a:Perfil {name: "Arthur"}), (m:Perfil {name: "Uncle Monty"}), (s)-[:AMIZADE]->(v),
(s)-[:AMIZADE]->(k), (s)-[:AMIZADE]->(m), (v)-[:AMIZADE]->(s), (v)-[:AMIZADE]->(k), (v)-[:AMIZADE]->(d), (v)-[:AMIZADE]->(m), (k)-[:AMIZADE]->(s),
(k)-[:AMIZADE]->(v), (d)-[:AMIZADE]->(m), (e)-[:AMIZADE]->(o), (o)-[:AMIZADE]->(s), (o)-[:AMIZADE]->(v), (o)-[:AMIZADE]->(k), (o)-[:AMIZADE]->(e),
(o)-[:AMIZADE]->(d), (p1:Post{texto: 'Socorro! Cond Olaf é um assassino!'}), (p2:Post{texto: 'Stehano é o Cond Olaf!'}), (v)-[:POSTED]->(p1),
(k)-[:POSTED]->(p2), (o)-[:COMMENTED{texto: "Não é Nada"}]->(p1), (k)-[:LIKED]->(p1), (d)-[:LIKED]->(p1), (s)-[:LIKED]->(p1), (v)-[:LIKED]->(p2);
```

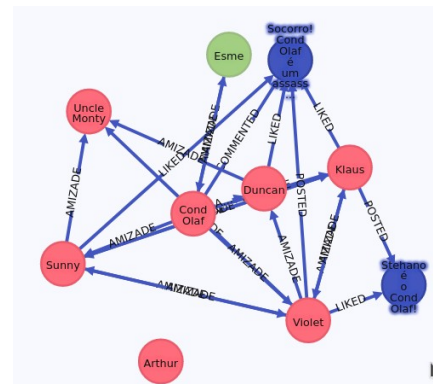


# Neo4J

- Popular o BD

```
CREATE (s:Perfil {name: "Sunny"}), (v:Perfil {name: "Violet"}), (k:Perfil {name: "Klaus"}), (o:Perfil {name: "Cond Olaf"}),
(d:Perfil {name: "Duncan"}), (e:Perfil {name: "Esme"}), (a:Perfil {name: "Arthur"}), (m:Perfil {name: "Uncle Monty"}), (s)-[:AMIZADE]->(v),
(s)-[:AMIZADE]->(k), (s)-[:AMIZADE]->(m), (v)-[:AMIZADE]->(s), (v)-[:AMIZADE]->(k), (v)-[:AMIZADE]->(d), (v)-[:AMIZADE]->(m), (k)-[:AMIZADE]->(s),
(k)-[:AMIZADE]->(v), (d)-[:AMIZADE]->(m), (e)-[:AMIZADE]->(o), (o)-[:AMIZADE]->(s), (o)-[:AMIZADE]->(v), (o)-[:AMIZADE]->(k), (o)-[:AMIZADE]->(e),
(o)-[:AMIZADE]->(d), (p1:Post{texto: 'Socorro! Cond Olaf é um assassino!'}), (p2:Post{texto: 'Stehano é o Cond Olaf!'}), (v)-[:POSTED]->(p1),
(k)-[:POSTED]->(p2), (o)-[:COMMENTED{texto: "Não é Nada"}]->(p1), (k)-[:LIKED]->(p1), (d)-[:LIKED]->(p1), (s)-[:LIKED]->(p1), (v)-[:LIKED]->(p2);
```

<https://drive.google.com/file/d/19qNGIbzM2JbvNbxguXZpLuw5k29Mndi/view?usp=sharing>



# Neo4J

- Consultas
  - Todos os nós da base:  
**MATCH** (n) **RETURN** n
  - Todos os nós que tenham uma propriedade:  
**MATCH** (a:Perfil)  
**RETURN** a.name
  - Qualquer nó relacionado independente da direção:  
**MATCH** (Perfil{name: "Cond Olaf" })--(c:Perfil)  
**RETURN** c.name
  - Qualquer nó relacionado com uma aresta saindo do nó de match:  
**MATCH** (Perfil{name: "Cond Olaf" })-->(c:Perfil)  
**RETURN** c.name

# Neo4J

- Consultas
    - Qualquer nó relacionado com uma aresta de tipo específico
      - **MATCH** (Perfil { name: “Neo” })-[:AMIZADE]-(c:Perfil)  
**RETURN** c.name
    - Match com múltiplas relações:  
**MATCH** (Perfil { name: “Sunny” })-[:Posted|:liked]-(c:Post)  
**RETURN** c.texto
    - Limitar o número de hops:  
**MATCH** (n:Perfil { name: 'Sunny' })-[:AMIZADE]-()-[:AMIZADE]-(c:Perfil) **RETURN** c.name
- MATCH** (n:Perfil { name: “Sunny” })-[:AMIZADE\*1..2]-(c:Perfil)  
**RETURN** c.name

# Neo4J



- Exercícios
  - Listar todos os amigos do Cond Olaf
  - Listar os amigos dos amigos da “Violet”
  - Buscar os posts feitos pelo Klaus
  - Buscar os posts comentados pelo Cond Olaf
  - Quantos são os que consideram Cond Olaf como amigo?

**SBBBD**  
2021



**UNIVERSIDADE FEDERAL  
DE SANTA CATARINA**

**GBD**  
INE-UFSC

# **Minicurso: Projeto de Bancos de Dados NoSQL (Prática)**

**Angelo Augusto Frozza**  
**Geomar André Schreiner**  
**Ronaldo dos Santos Mello**



**GBD/INE/CTC/UFSC**

