

FIAP GRADUAÇÃO

Architecture Analytics & NoSQL

Neo4J

Milton Goya

Versão 01 – fevereiro de 24

2

NoSQL BD Grapho Neo4J



Principais Bancos Orientados a Graphos

☐ include secondary database models

40 systems in ranking, November 2023

Rank			DBMS	Database Model	Score		
Nov 2023	Oct 2023	Nov 2022			Nov 2023	Oct 2023	Nov 2022
1.	1.	1.	Neo4j	Graph	49.70	+1.26	-7.60
2.	2.	2.	Microsoft Azure Cosmos DB	Multi-model	34.11	-0.18	-5.64
3.	3.	3.	Virtuoso	Multi-model	5.62	+0.19	-0.22
4.	5.	4.	ArangoDB	Multi-model	4.54	+0.27	-1.30
5.	4.	5.	OrientDB	Multi-model	3.78	-0.49	-1.26
6.	6.	19.	Memgraph	Graph	3.09	+0.29	+2.18
7.	8.	8.	GraphDB	Multi-model	2.75	+0.16	+0.29
8.	7.	16.	NebulaGraph	Graph	2.54	-0.24	+1.27
9.	9.	6.	Amazon Neptune	Multi-model	2.49	-0.06	-0.60
10.	11.	10.	Stardog	Multi-model	2.30	+0.04	+0.54

- <https://db-engines.com/en/ranking/graph+dbms>

Introdução



A ideia de Grafo surgiu independente das diversas áreas de conhecimento, no entanto, é considerada como uma área da matemática aplicada. A mais antiga menção sobre o assunto ocorreu no trabalho de Euler (pronuncia-se Óiler).



Introdução

a



A teoria surgiu pela primeira vez em 1736 e foi criada por Leonhard Euler para resolver um problema chamado de “As 7 pontes de Königsberg” (atual Kaliningrado). Seis delas interligavam duas ilhas às margens do Rio Pregel e uma que fazia a ligação entre as duas ilhas. O problema consistia na seguinte questão: como seria possível fazer um passeio a pé pela cidade, de forma a passar uma única vez por cada uma das sete pontes e retornar ao ponto de partida?



Introdução

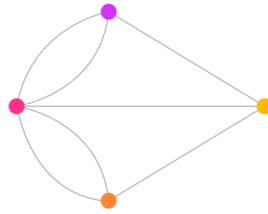
a



Euler focou apenas no problema removendo todos os respectivos detalhes geométricos (distância, tamanho das pontes, formas etc.) e apresentou uma solução que, além de provar que não era possível passar apenas uma vez por ponte, mostrou como a forma de solução poderia ser aplicada a outros problemas semelhantes. Assim surgiu a teoria dos grafos.



Introdução

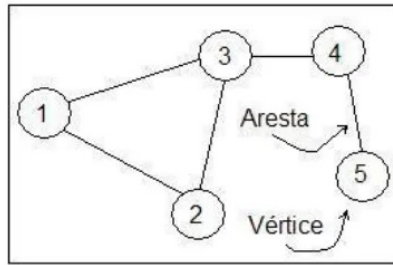


Um banco de dados grafo é um banco de dados projetado para tratar os relacionamentos entre dados como o principal aspecto no modelo de dados, ou seja, são vértices unidos por arestas (linhas).

Essa composição do grafo foi adaptada e trazida para a ótica do banco de dados, o que mudou um pouco os conceitos para nós e relacionamentos. Em uma base de grafos, o **nó** armazena as informações de uma **entidade**, e os **relacionamentos**, as informações de **como** as entidades se relacionam.



Introdução



Vértice (ou Nó):

- Equivalente a uma entidade em um banco de dados relacional.
- Representa entidades e objetos, como um usuário, uma cidade, ou um produto.
- Pode conter propriedades (pares de chave-valor) que descrevem os atributos do vértice.

Aresta (ou Relação):

- Equivalente a um relacionamento em um banco de dados relacional.
- Representa como os vértices estão conectados entre si.
- As arestas são direcionadas e podem conter propriedades próprias, o que oferece um contexto adicional sobre a natureza do relacionamento (por exemplo, peso, tipo de relação, etc.).

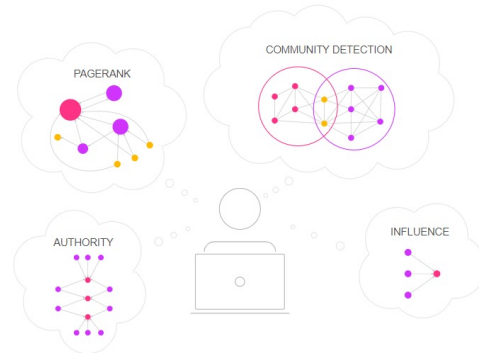
Propriedade (ou Atributo)

- São atributos associados tanto aos nós quanto aos relacionamentos, fornecendo informações detalhadas e contextuais



■ Aplicações do Neo4j:

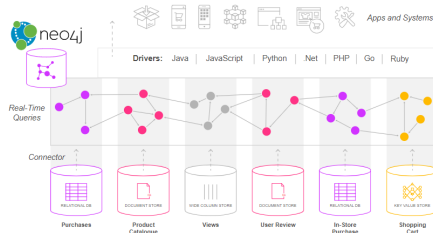
1. Redes Sociais
2. Detecção de Fraude
3. Recomendação de Produtos
4. Gerenciamento de Redes de TI
5. Bioinformática
6. Logística e Cadeia de Suprimentos
7. Análise de Dados Conectados



Os modelos de banco de dados grafo são aplicados em áreas nas quais as informações sobre a interconectividade ou topologia de dados sejam mais importantes ou tão importantes quanto os próprios dados.



Neo4J



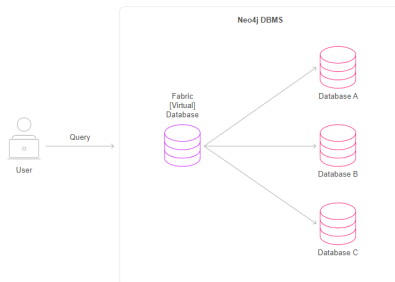
Neo4j é um banco de dados com uma estrutura física de arquivos organizados em um diretório ou pasta, que tem o mesmo nome do banco de dados. Em termos lógicos, um banco de dados é um contêiner para um ou mais grafos.

Uma instalação padrão do Neo4j 4.1 contém dois bancos de dados:

- **system** – o banco de dados do sistema, contendo metadados no DBMS e na configuração de segurança.
- **neo4j** – o banco de dados padrão, um único banco de dados para dados do usuário. Ele tem um nome padrão de neo4j. Um nome diferente pode ser configurado antes de iniciar o Neo4j pela primeira vez.

Drivers para linguagens de programação populares, incluindo Java, JavaScript, .NET, Python e muito mais.

Neo4J Fabric



O Neo4j Fabric é um módulo para permitir consultar vários grafos dentro da mesma transação com Cypher, ou seja, realiza a federação de dados – uma visão unificada dos dados locais e distribuídos, acessíveis por meio de uma única conexão de cliente e sessão do usuário.

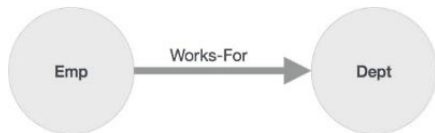
- Cypher: uma linguagem de consulta declarativa semelhante ao SQL, mas otimizada para grafos. Agora utilizada por outros bancos de dados, como o SAP HANA Graph e Redis, por meio do projeto openCypher.
- Tempo para retornar os percursos constantes em grandes grafos, com consultas tanto em profundidade quanto em largura, devido à representação eficiente de nós e relacionamentos. Permite expansão para bilhões de nós com requisitos de hardware considerados moderados.
- Esquema flexível para propriedades que pode se adaptar ao longo do tempo.

Modelo de Dados



- **Nós** são as entidades no gráfico.
 - Eles podem conter qualquer número de atributos (pares de valor-chave) chamados propriedades.
 - Os nós podem ser marcados com marcadores representando suas diferentes funções no seu domínio.
 - Além de contextualizar propriedades de nó e relacionamento, os rótulos também podem servir para anexar metadados – informações de índice ou restrição – a determinados nós.

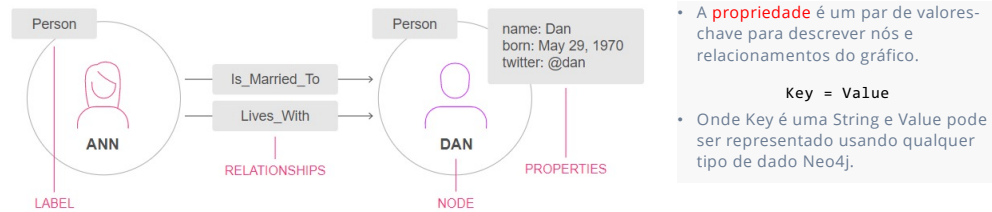
Modelo de Dados



- Emp e Dept são dois nós diferentes. "WORKS_FOR" é um relacionamento entre os nós Emp e Dept.
- A marca de seta de Emp para Dept, esse relacionamento descreve – Emp WORKS_FOR Dept
- "Emp" é um nó inicial e "Dept" é um nó final.

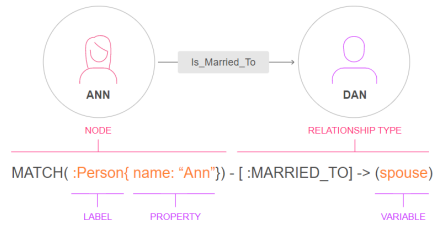
- **Relacionamentos** fornecem conexões dirigidas, nomeadas semanticamente relevantes entre duas entidades de nó (por exemplo, Employee WORKS_FOR Company).
 - Um relacionamento sempre tem uma direção, um tipo, um nó inicial e um nó final. Como os nós, os relacionamentos também podem ter propriedades. Na maioria dos casos, os relacionamentos têm propriedades quantitativas, como pesos, custos, distâncias, classificações, intervalos de tempo ou intensidades.
 - Como os relacionamentos são armazenados eficientemente, dois nós podem compartilhar qualquer número ou tipo de relacionamento sem sacrificar o desempenho. Observe que, embora sejam direcionados, os relacionamentos sempre podem ser navegados eficientemente em qualquer direção.

Modelo de Dados



- Os nós são os principais elementos de dados.
- Os nós estão conectados a outros nós por meio de relacionamentos.
- Os nós podem ter uma ou mais propriedades (isto é, atributos armazenados como pares chave/valor).
- Os nós têm um ou mais rótulos (label) que descrevem seu papel no gráfico.
- Relacionamentos conectam dois nós.
- Relacionamentos são direcionais.
- Os nós podem ter relacionamentos múltiplos, até recursivos.
- Relacionamentos podem ter uma ou mais propriedades (isto é, atributos armazenados como pares chave / valor).

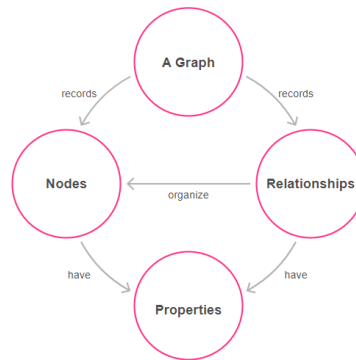
Modelo de Dados



- O **rótulo (label)** associa um nome comum a um conjunto de nós ou relacionamentos.
- Um nó ou relacionamento pode conter um ou mais rótulos.
- Podemos criar novos rótulos para nós ou relacionamentos existentes.
- Podemos remover os rótulos existentes dos nós ou relacionamentos existentes.

- Os **rótulos** são usados para agrupar nós em conjuntos.
 - Um nó pode ter vários rótulos.
 - Os rótulos são indexados para acelerar a localização de nós no gráfico.
 - Índices de rótulos nativos são otimizados para velocidade.
- As **propriedades**:
 - Propriedades são valores nomeados em que o nome (ou chave) é uma string.
 - Propriedades podem ser indexadas e restringidas.
 - Índices compostos podem ser criados a partir de múltiplas propriedades.

Modelo de Dados



- Um grafo possui nós e arestas denominados, no Neo4J, nodes e relationships, respectivamente. Tanto os nós como as arestas possuem propriedades (properties).
- Cypher é uma linguagem de consulta declarativa do Neo4j que permite consultas e atualizações expressivas e eficientes dos dados no grafo.

Modelo de Dados

- Sintaxe para criar um nó:

```
CREATE (nomeDoNo:Label {propriedade1: valor1, propriedade2: valor2, ...})
```

- Onde:
 - nomeDoNo: O nome do nó, que pode ser qualquer identificador válido.
 - Label: Um rótulo opcional que pode ser usado para agrupar nós com características semelhantes.
 - propriedade1, propriedade2, ...: As propriedades associadas ao nó, cada uma com um valor específico.

- Exemplo:

```
CREATE (u1:Usuário {Nome: "Beatrice", Id: 1 })  
CREATE (u2:Usuário {Nome: "Mateus", Id: 2 })  
CREATE (u3:Usuário {Nome: "Pedro", Id: 3 })  
CREATE (u4:Usuário {Nome: "Wellynton", Id: 4 })
```

Criando Nós

```
CREATE (p:Pessoa {nome: 'João', idade: 22, genero: 'masculino'})  
CALL db.schema.visualization()
```

- CREATE é o comando usado para criar algo no banco de dados.
- (p:Pessoa { ... }) cria um nó com o rótulo Pessoa. O p é um identificador que você pode usar para se referir a esse nó em comandos posteriores dentro da mesma transação.
- {nome: 'João', idade: 22, genero: 'masculino'} são as propriedades do nó, onde você define um conjunto de pares chave-valor que armazenam os dados do nó.

```
CREATE (m:Pessoa {nome: 'Maria', genero: 'feminino'})  
CREATE (j:Pessoa {nome: 'Joselito', idade: 22})  
RETURN labels(m) AS Maria, labels(j) AS Joselito  
CALL db.schema.visualization()
```

Criando Nós com Relacionamento

```
CREATE (severino:Pessoa {nome: 'Severino', idade: 60})
CREATE (severina:Pessoa {nome: 'Severina', idade: 60})
CREATE (severino)-[r:LOVES]->(severina)
RETURN r
CALL db.schema.visualization()
```

- CREATE (severino:Pessoa {nome: 'Severino', idade: 60}) cria o nó para Severino com o rótulo Pessoa.
- CREATE (severina:Pessoa {nome: 'Severina', idade: 60}) cria o nó para Severina com o rótulo Pessoa.
- CREATE (severino)-[r:LOVES]->(severina) cria uma relação de "LOVES" de Severino para Severina. A relação é direcionada, indicando que Severino ama Severina.
- RETURN r retorna a relação criada.

```
CREATE (severino:Pessoa {nome: 'Severino', idade: 60}),
      (severina:Pessoa {nome: 'Severina', idade: 60}),
      (severino)-[r:LOVES]->(severina)
RETURN r
CALL db.schema.visualization()
```

Criando Relacionamento entre nós existentes

```
MATCH (a:Pessoa), (b:Pessoa)
WHERE a.nome = 'Severino' AND b.nome = 'Severina'
CREATE (b)-[r:LOVES]->(a)
RETURN r
CALL db.schema.visualization()
```

MATCH (a:Pessoa), (b:Pessoa): Este comando procura na base de dados por dois nós distintos, ambos com o rótulo Pessoa. As variáveis a e b são alias que serão usadas para referenciar estes nós após serem encontrados.

WHERE a.nome = 'Severino' AND b.nome = 'Severina': Este é um filtro que se aplica aos resultados do comando MATCH. Restringe os nós encontrados para aqueles onde o nó referenciado por a tem uma propriedade nome com o valor 'Severino' e o nó referenciado por b tem uma propriedade nome com o valor 'Severina'.

CREATE (b)-[r:LOVES]->(a): Após encontrar os nós que correspondem aos critérios definidos, esta parte da consulta cria um relacionamento direcionado do nó b (Severina) para o nó a (Severino), com o tipo de relacionamento LOVES. A variável r é um alias para referenciar esse relacionamento.

RETURN r: Por fim, esta parte da consulta retorna o relacionamento recém-criado. Isso permite que você veja as propriedades do relacionamento ou simplesmente confirme que ele foi criado com sucesso.

Criando Relacionamento entre nós existentes

```
MATCH (m:Pessoa), (j:Pessoa)
WHERE m.nome = 'Maria' AND j.nome = 'João'
CREATE (m)-[r:LOVES]->(j)
RETURN r
```

```
MATCH (joselito:Pessoa), (maria:Pessoa)
WHERE joselito.nome = 'Joselito' AND maria.nome = 'Maria'
CREATE (joselito)-[r:LOVES]->(maria)
RETURN r
```

```
MATCH (joselito:Pessoa {nome: 'Joselito'})-[r:LOVES]->(maria:Pessoa
{nome: 'Maria'})
DELETE r
```

Criando nós e Relacionamentos

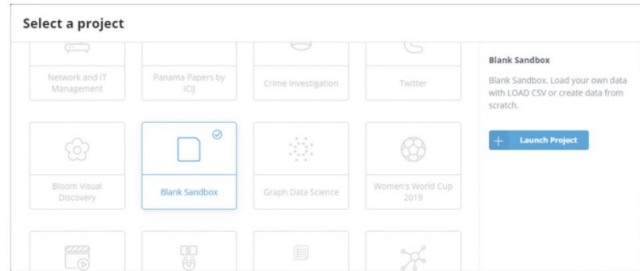
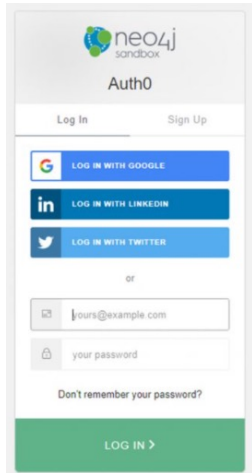
```
// Criar nós  
CREATE (e:Empregado {nome: 'João Silva', cargo: 'Analista de  
Sistemas'})  
CREATE (d:Departamento {nome: 'TI', local: 'Sede Central'})
```

```
// Criar relação  
MATCH (e:Empregado), (d:Departamento)  
WHERE e.nome = 'João Silva' AND d.nome = 'TI'  
CREATE (e)-[r:TRABALHA_PARA]->(d)
```

```
// Retorna gráfico  
CALL db.schema.visualization()
```

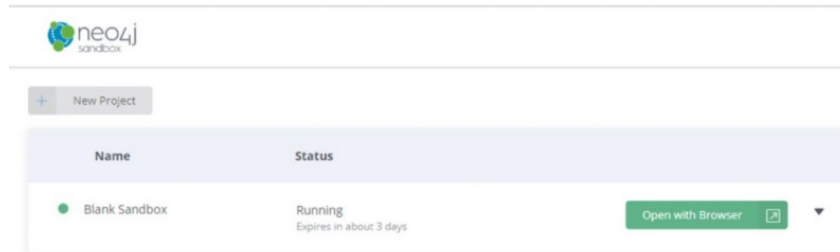

Acesso ao Neo4J

- Podemos optar pela versão na nuvem <https://sandbox.neo4j.com/>
- Ou realizar uma instalação local <https://neo4j.com/download/>.



Selecionar Blank Sandbox no Sandbox Neo4j.

Acesso ao Neo4J



Nosso projeto Blank Sandbox foi criado e temos três dias para realizar comandos nesse ambiente. Para isso, abrimos uma interface gráfica no browser (Open with Browser).

Neo4J Desktop

Download Neo4j Desktop

Experience Neo4j on Your Desktop

Free. Get Started Today.

Download 

Includes Neo4j Enterprise 5.13.0 for Developers
[Learn more](#) | [System Requirements](#)

<https://neo4j.com/download/>

Neo4J Desktop

Get Started Now

Please fill out this form to begin your download

Brazil

By downloading you agree to the [Neo4j License Agreement for Neo4j Desktop Software](#).

Download Desktop


The information you provide will be used in accordance with the terms of our [Privacy Policy](#)

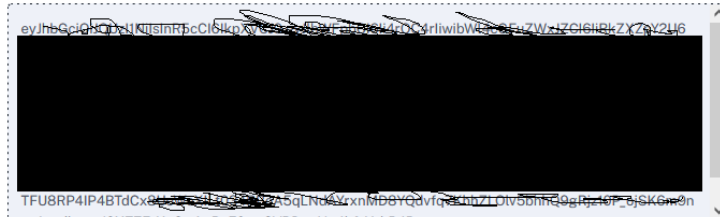
Para instalação local, forneça seus dados, o instalador será baixado

Neo4J Desktop

Neo4j Desktop Activation Key

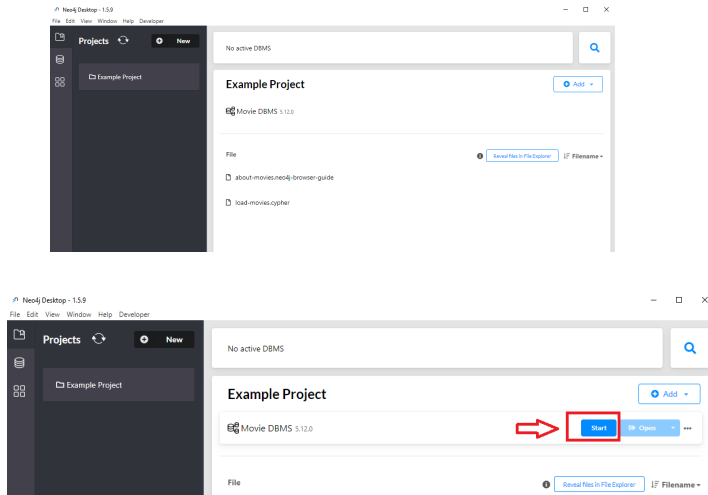
Use this key to activate your copy of Neo4j Desktop for use.

 **Copy to clipboard**

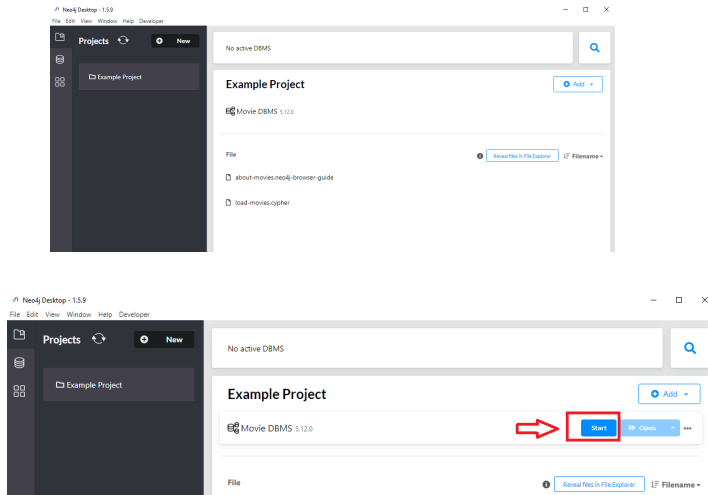


Copie sua "Activation Key" para sua versão desktop e realize a instalação.

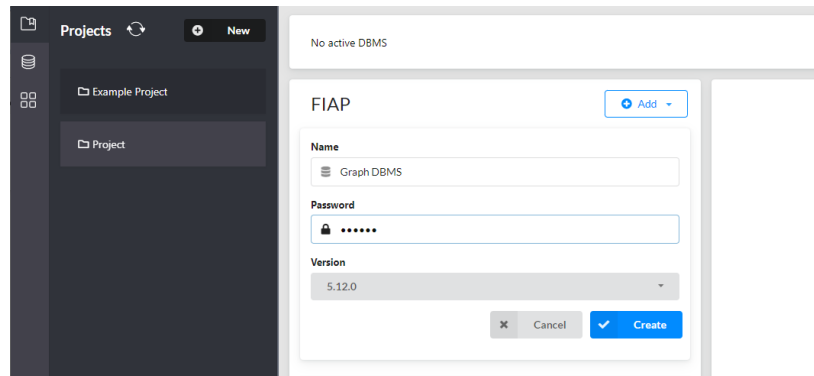
Neo4J Desktop



Neo4J Desktop



Neo4J Desktop



Relações do filme Matrix

:play movie-graph

CALL db.schema.visualization()

CREATE (TheMatrix:Movie {title:'The Matrix', released:1999, tagline:'Welcome to the Real World'})

CREATE (Keanu:Person {name:'Keanu Reeves', born:1964})

CREATE
(Keanu)-[:ACTED_IN {roles:['Neo']}]>(TheMatrix)

MATCH (Keanu:Person),(TheMatrix:Movie)
WHERE Keanu.name='Keanu Reeves' AND TheMatrix.title='The Matrix'

CREATE
(Keanu)-[:ACTED_IN {roles:['Neo']}]>(TheMatrix)
Return Keanu, TheMatrix