# BANCO DE DADOS NOSQL

AULA 5

Prof. Alex Mateus Porn

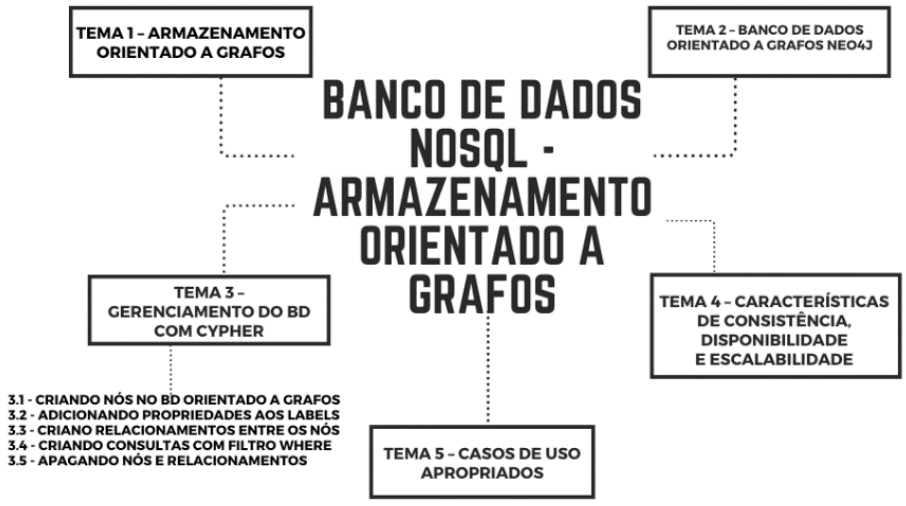
## CONVERSA INICIAL

Nesta aula, abordaremos o tema de banco de dados orientados a grafos. Temos como maior objetivo desta aula introduzir os principais conceitos sobre o modo de armazenamento e gerenciamento de dados orientados a grafos, compreendendo principalmente a estrutura física de armazenamento desses bancos e as características de relacionamento entre os dados.

Iniciaremos com uma abordagem geral sobre a estrutura de armazenamento orientada a grafos, incluindo as principais definições e conceitos desse tipo de banco de dados. Você aprenderá como os dados são estruturados no tipo orientado a grafos, e como são armazenados em nós (ou *vértices,* como também são chamados). Ao longo do estudo desses conceitos exclusivos do modelo orientado a grafos, faremos comparações com o modelo de dados relacional.

Após a compreensão desse modelo de dados, aplicaremos todos os conceitos aprendidos de forma prática no SGBD Neo4j, de modo que será possível compreender todo o processo de criação do banco de dados, incluindo inserção, edição e criação de consultas aos dados.

Figura 1 – Roteiro da aula




## TEMA 1 – ARMAZENAMENTO ORIENTADO A GRAFOS

Já conhecemos quatro tipos de armazenamento de dados NoSQL: orientado a chave-valor; orientado a documentos; orientado a colunas; além do

modelo estudado nesta aula, orientado a grafos. Este último é provavelmente o mais especializado. Conforme Marquesone (2017, p. 54):

> Diferente dos outros modelos, em vez dos dados serem modelados utilizando um formato de linhas e colunas, eles possuem uma estrutura definida na teoria dos grafos, usando vértices e arestas para armazenar os dados dos itens coletados (como pessoas, cidades, produtos e dispositivos) e os relacionamentos entre esses dados, respectivamente.

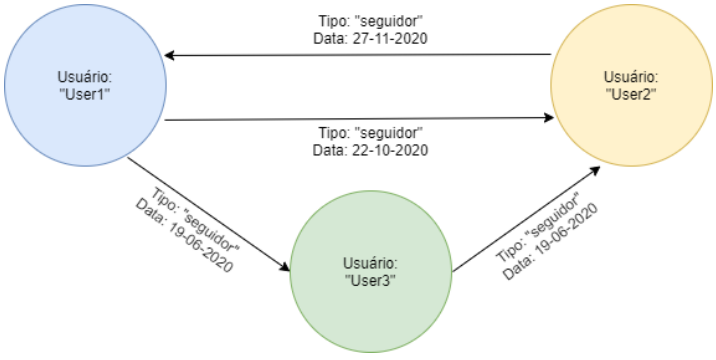
Seguindo nessa análise, Marquesone (2017, p. 55) ainda afirma que o modelo orientado a grafos oferece maior desempenho em aplicações que precisam traçar os caminhos existentes nos relacionamentos entre os dados. Por exemplo, as aplicações que precisam identificar como que um conjunto de amigos está conectado em uma rede, ou descobrir a melhor rota para chegar a determinado local em menor tempo.

Nesse contexto, em que a descoberta da forma como os dados estão relacionados é mais importante do que os dados em si, podemos citar como exemplo os relacionamentos entre os usuários de uma rede social, ou entre os usuários de uma aplicação comercial. Nesse tipo de armazenamento de dados, além das informações armazenadas sobre cada usuário, são também armazenadas informações sobre a ligação entre eles.

> Esse mesmo tipo de informação pode ser usada em toda a rede de usuários, possibilitando a criação de soluções baseadas nessa análise, tais como a recomendação de amigos com base na rede de relacionamentos. Em situações como essa, com foco no relacionamento dos dados, é que o banco de dados orientado a grafos é recomendado. (Marquesone, 2017, p. 54)

A Figura 2 apresenta um exemplo da estrutura de um banco de dados orientado a grafos, que armazena o relacionamento entre os usuários de uma rede social.

Figura 2 – Exemplo da estrutura de um banco de dados orientado a grafos




Fonte: Elaborado com base em Marquesone, 2017, p. 54.

De acordo com o exemplo apresentado na figura, podemos identificar que o usuário "User1" é um seguidor do usuário "User2", que também é seu seguidor. Por outro lado, “User1” é seguidor de “User3”, que é seguidor apenas de “User2”.

Com vistas a manter o armazenamento dos relacionamentos entre os registros, Marquesone (2017, p. 55) destaca que outros modelos de armazenamento, até mesmo o relacional, também são capazes de realizar consultas sobre os relacionamentos entre os itens armazenados. Porém, em situações com milhões de relacionamentos, essa consulta pode se tornar muito complexa, resultando em baixo desempenho.

Seguindo essa abordagem, vale destacar a afirmação de Hecht e Jablonski (2011), que para situações de gerenciamento eficiente de dados fortemente vinculados, os bancos de dados orientados a grafos são especializados: “Aplicativos baseados em dados com muitos relacionamentos são mais adequados para bancos de dados orientados a grafos, uma vez que operações de alto custo, como junções recursivas, podem ser substituídas por travessias eficientes”.

Em relação à estrutura, os sistemas de bancos de dados em grafos modelam os dados por meio de vértices e arestas, facilitando a modelagem de contextos complexos, e definindo naturalmente relações existentes entre as entidades de uma base. Nessa categoria, os sistemas podem ser classificados como nativos ou não-nativos (Penteado et al., 2014):

- **Nativos:** usam listas de adjacências. Em uma lista de adjacência, cada vértice mantém referências diretas para seus vértices adjacentes, formando uma espécie de micro índice para os vértices próximos. A estrutura de grafo é considerada tanto no armazenamento físico dos dados quanto no processamento de consultas.
- **Não-nativos:** modelam logicamente seus dados como grafos, porém armazenam os dados por meio de outros modelos. Alguns sistemas armazenam suas triplas em tabelas relacionais, outros modelos armazenam o grafo fisicamente em uma estrutura chave-valor. Por exemplo, por meio do modelo relacional, as relações de triplas vértice-aresta-vértice em um grafo são armazenadas como tuplas em tabelas. Esse tipo de composição é prejudicial para o desempenho de consultas, quando diversas junções são necessárias para executar uma consulta complexa envolvendo diversas triplas.

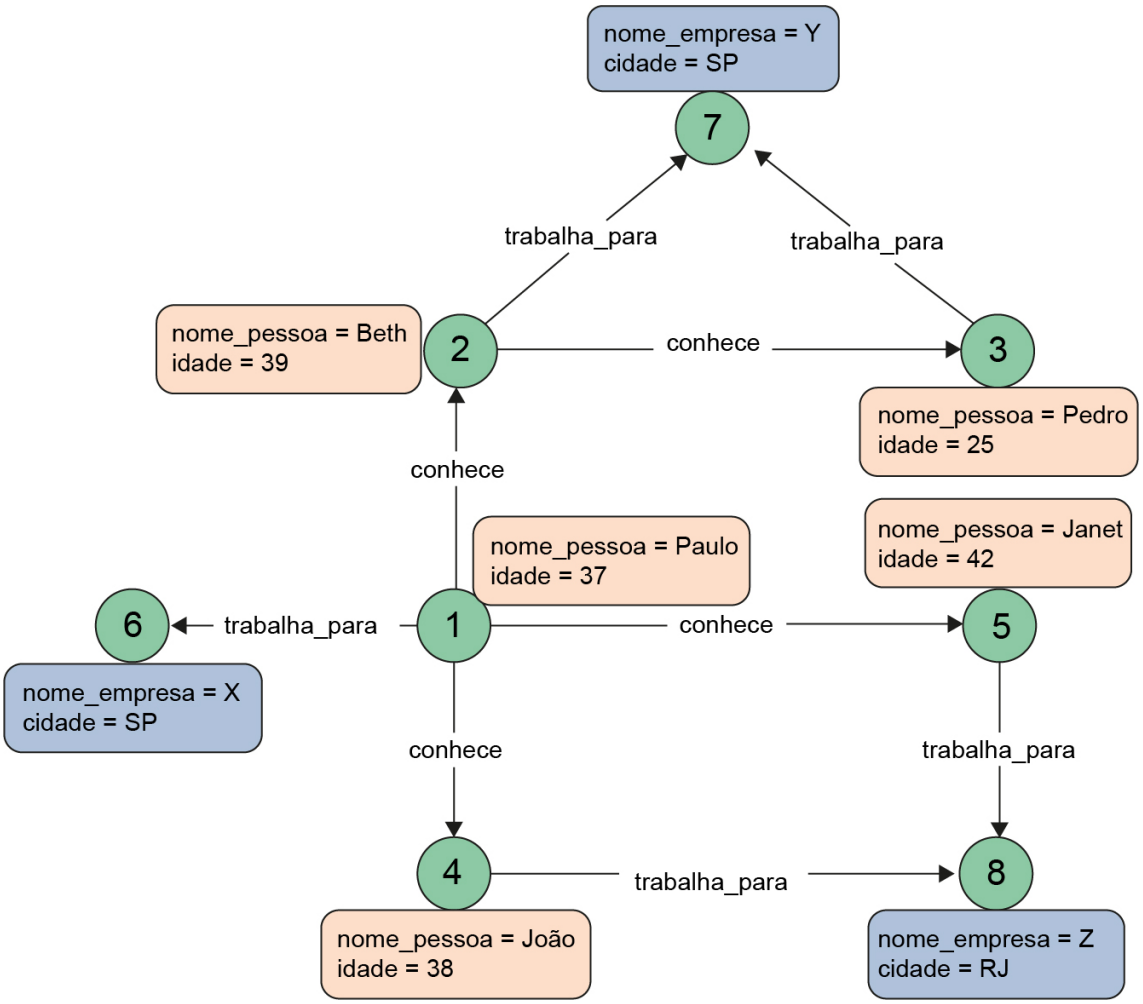
Desse modo, ao implementar um banco de dados orientado a grafos nativo ou não-nativo, um modelo deve ser implementado para o armazenamento físico dos dados, representados por meio de vértices e arestas no modelo lógico (Penteado et al., 2014). Por exemplo, as listas de adjacências podem ser armazenadas em um repositório por meio do modelo chave-valor, em que a chave identifica um vértice e o valor referencia a lista de adjacências.

Ao modelar uma base de dados, diferentes formas podem ser definidas dependendo do modelo de grafo escolhido. De acordo com Penteado et al. (2014), um grafo pode seguir um dos seguintes modelos:

- **Grafo simples-relacional:** modelo bem simples e limitado, em que todos os vértices denotam o mesmo tipo de objeto, e todas as arestas denotam o mesmo tipo de relacionamento.
- **Grafo multi-relacional:** permite um conjunto variado de tipos de objetos e de relacionamentos, possibilitando múltiplas relações e um maior poder de modelagem.
- **Grafo de propriedades:** grafo multi-relacional com atributos e arestas direcionadas. Uma aresta pode ser direcionada e/ou rotulada e/ou valorada com um peso em um modelo. Adicionalmente, arestas e vértices podem ter propriedades com valores associados.

Seguindo essa abordagem de Penteado et al. (2014), o modelo mais utilizado é o grafo de propriedades. A Figura 3 apresenta um exemplo de um grafo de propriedades no contexto de uma rede social corporativa. Nesse exemplo, os vértices que representam as pessoas apresentam as propriedades “nome” e “idade”, enquanto os vértices que representam as empresas têm as propriedades “nome” e “cidade”. As arestas representam as relações “um empregado trabalha para uma empresa” e “uma pessoa conhece outra pessoa”.

Figura 3 – Exemplo de um grafo de propriedades.



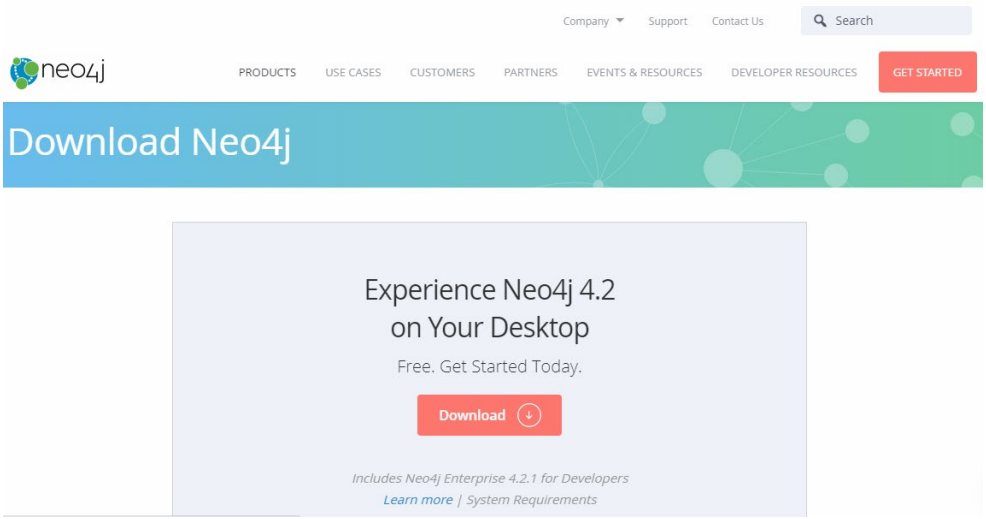

Fonte: Elaborado com base em Penteado, 2014.

## TEMA 2 – BANCO DE DADOS ORIENTADO A GRAFOS NEO4J

O Neo4j é um sistema gerenciador de bancos de dados orientado a grafos, e utiliza o modelo de grafos de propriedades. Esse SGBD teve a sua primeira versão lançada no ano de 2010. Foi implementado na linguagem de programação Java, tanto com uma versão de licenciamento aberta quanto proprietária. Assim como vimos em aulas anteriores sobre o SGBD HBase, que pode ser implementado de forma local ou distribuída, o Neo4j possibilita dois modos de implementação. Nesta aula, estudaremos o Neo4j no seu formato de licenciamento aberto e implementado de forma local, apesar de, sempre que possível, mencionarmos suas características distribuídas e de replicação.

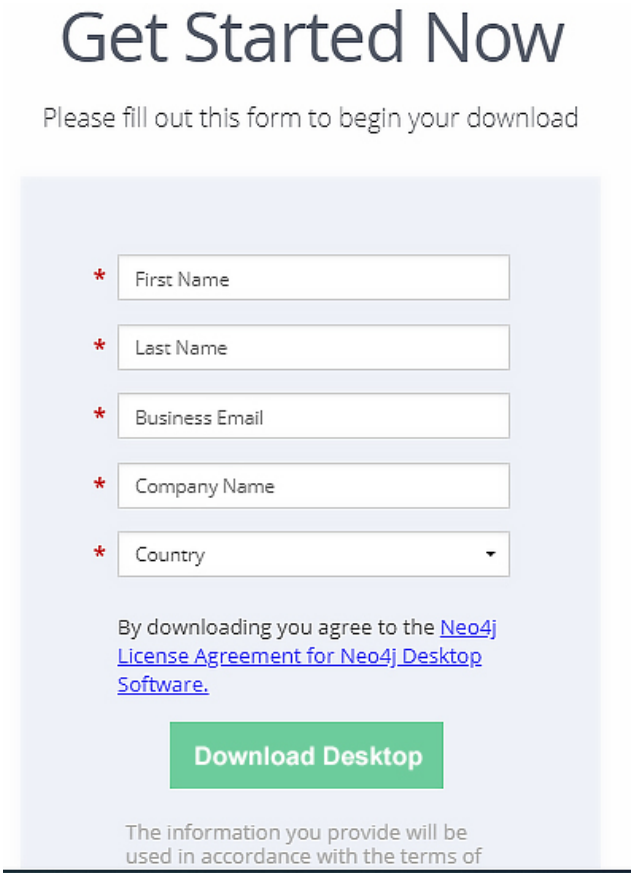
Para instalar o Neo4j, primeiramente verifique se está usando a versão mais atual do Java JDK. Em seguida, devemos fazer o download do Neo4j (disponível em: <https://neo4j.com/download/>).

Figura 4 – Tela de download do Neo4j



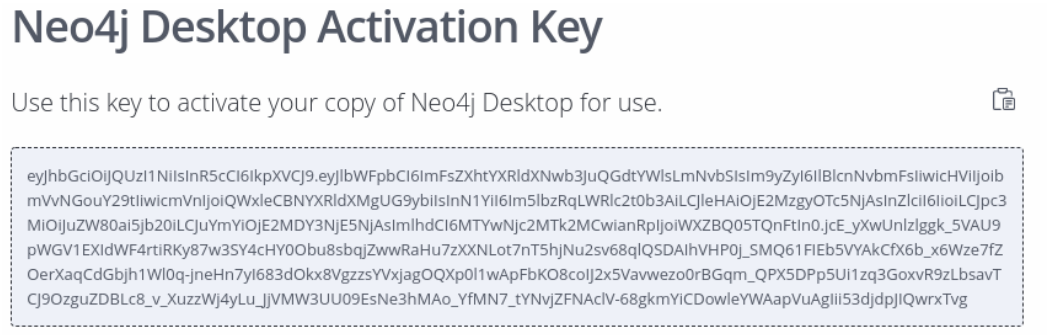

Conforme podemos ver na Figura 4, nesta aula abordaremos a versão 4.2 do Neo4j, que neste momento é a versão mais atual. Após clicar no botão *Download*, o próximo passo o levará a preencher um formulário com os dados pessoais para registro no Neo4j, conforme apresentado na Figura 5.

Figura 5 – Registro no Neo4j



Após o preenchimento do cadastro, basta clicar no botão *Download Desktop*, destacado em verde na Figura 5, e aguardar o download do SGBD Neo4j. Ao iniciar o download, será aberta uma nova janela, contendo uma chave de ativação do Neo4j, conforme mostrado na Figura 6. Copie essa chave clicando no ícone da “prancheta”, localizado no canto direito da figura, ou selecione toda a chave e pressione as teclas CTRL + C. Em seguida, cole o conteúdo em um arquivo de texto e salve em seu computador. Essa chave será necessária durante a configuração do SGBD.

Figura 6 – Chave de ativação do Neo4j.

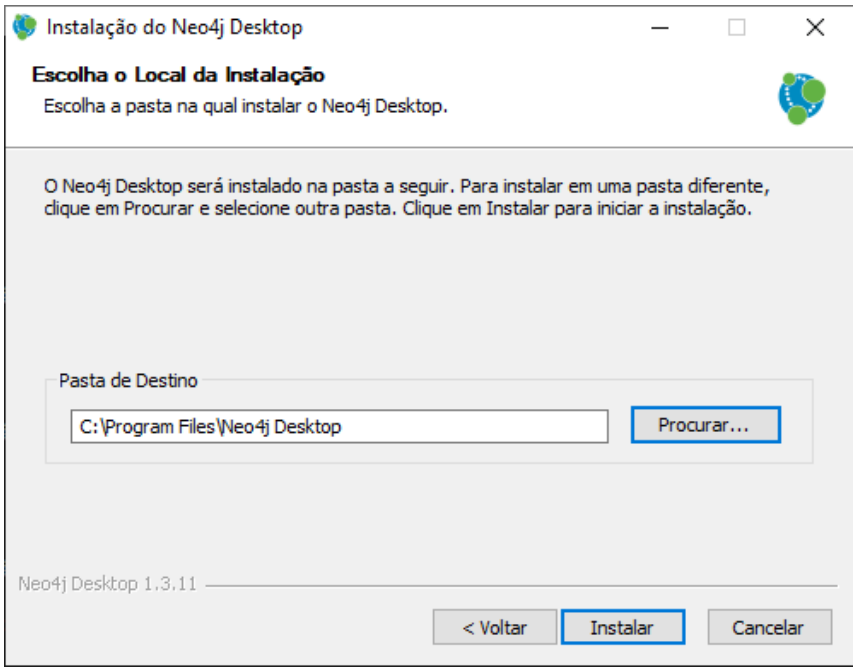



Após finalizado o download, precisamos dar um duplo clique para executar o arquivo de instalação. Recomenda-se não instalar o Neo4j em diretórios que contenham “espaço” no nome, como por exemplo o diretório “Arquivos de Programas” do Windows. Neste caso, criaremos uma pasta chamada Neo4jDesktop dentro da unidade C: no Windows. Nesta pasta, ficarão os arquivos do Neo4j. Também criaremos outra pasta chamada “Data” para armazenamento dos dados. Portanto, criaremos os seguintes diretórios:

- C:\Neo4jDesktop.
- C:\Neo4jDesktop\Data.

Após iniciada a instalação, atentar-se para alterar o local de instalação do Neo4j para o novo diretório Neo4jDesktop que acabamos de criar. Por padrão no sistema operacional Windows, é indicado o diretório “Arquivos de Programas”.

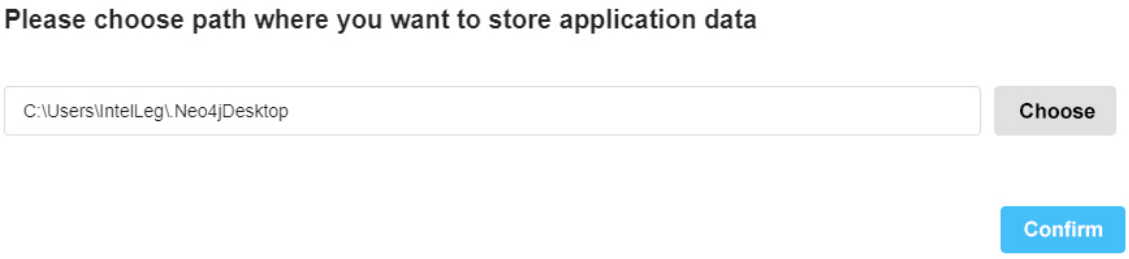
Figura 7 – Alteração do diretório de instalação do Neo4j




Ao chegar na tela para escolher o local de instalação do Neo4j, devemos clicar no botão “Procurar...” e selecionar a pasta “Neo4jDesktop”, que criamos no passo anterior. Em seguida, basta clicar em “Instalar”.

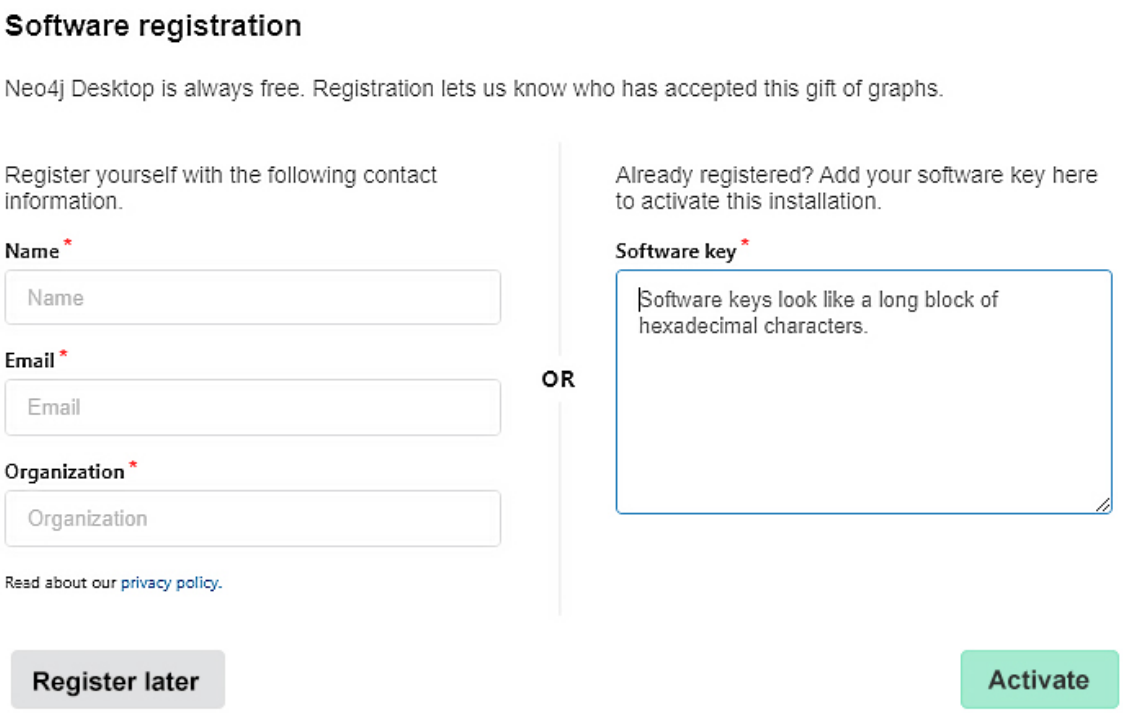
Ao inicializar o Neo4j pela primeira vez, será aberta a caixa de diálogo, conforme mostrada na Figura 8, solicitando o diretório onde serão armazenados os dados da aplicação. Neste ponto, devemos clicar no botão “*Choose*” e selecionar a pasta “Data”, que criamos dentro da pasta “Neo4jDesktop” na etapa anterior.

Figura 8 – Seleção do diretório para armazenamento dos dados da aplicação




Ao clicarmos no botão “*Confirm*”, destacado em azul na Figura 8, será aberta uma nova caixa de diálogo, conforme mostra a Figura 9, solicitando os dados para registro no Neo4j e a chave do software que obtivemos ao fazer o download do SGBD.

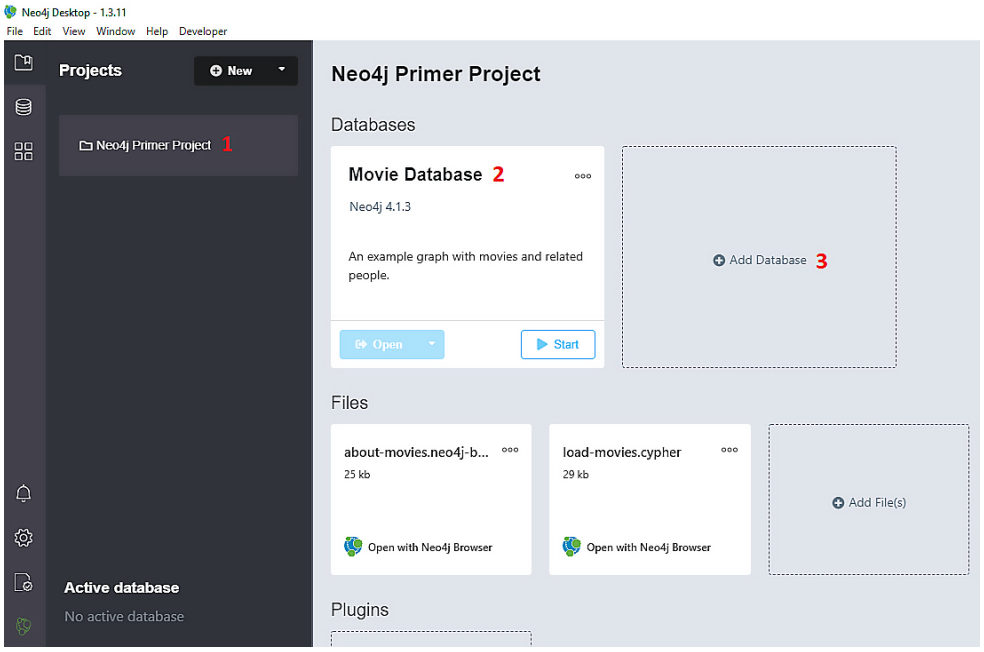
Figura 9 – Registro do SGBD Neo4j




Após o preenchimento do formulário, basta clicar no botão "*Activate*", destacado em verde na Figura 9, e aguardar o término da configuração do Neo4j. Outra alternativa é clicar no botão "*Register later*", mostrado no canto inferior esquerdo da figura, e finalizar a instalação do Neo4j sem realizar o registro.

Após o término da etapa anterior (realizar o registro ou inicializar sem ele), será carregada a interface gráfica inicial do Neo4j.

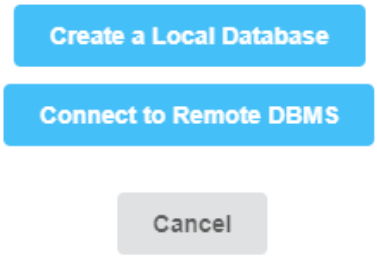
Figura 10 – Interface gráfica inicial do Neo4j



Como podemos ver na Figura 10, em destaque com o número 1 na cor vermelha no canto superior esquerdo da figura, vemos o nome do primeiro projeto iniciado com a instalação do Neo4j. Novos projetos podem ser criados clicando no botão "*New*". O nome do projeto inicial também pode ser alterado. No canto direito da Figura 10, destacado com o número 2, é disponibilizado um banco de dados de exemplo sobre filmes; destacado com o número 3, temos a ferramenta para criação de novos bancos de dados para o projeto selecionado.

Para criar uma base de dados para o projeto selecionado, basta clicar na opção "*Add Database*", marcada com o número 3 na Figura 10. Em seguida, haverá solicitação do tipo da nova base de dados, conforme mostra a Figura 11.

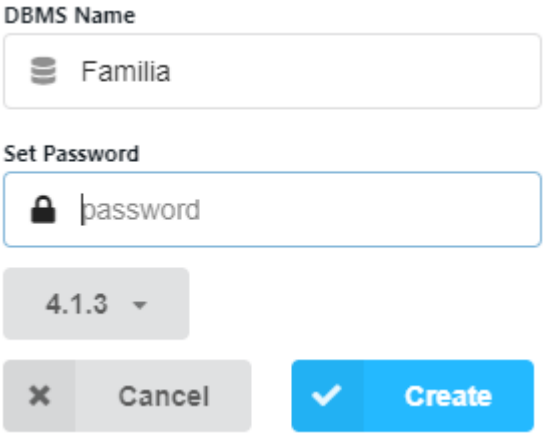
Figura 11 – Seleção do tipo da nova base de dados




- **Create a Local Database:** criará um novo banco de dados local;
- **Connect to Remote DMBS:** possibilitará a conexão em um banco de dados remoto.

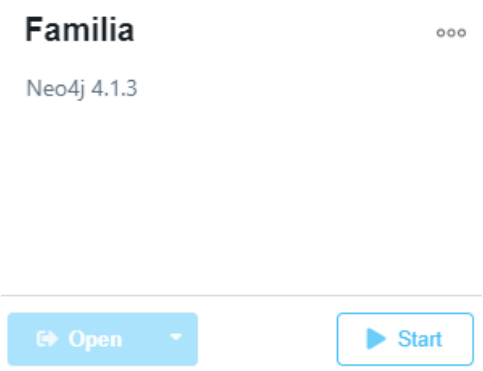
Nesta aula, estudaremos a criação e o gerenciamento de um banco de dados local. Para isso, devemos clicar no botão "*Create a Local Database*", conforme vimos na Figura 11, e em seguida informar o nome do novo banco de dados, criar uma senha para esse banco e finalizar clicando no botão "*Create*".

Figura 12 – Criação de um novo banco de dados local



Após criar a nova base de dados, precisamos inicializá-la, clicando no botão “*Start*”, conforme mostra a Figura 13.

Figura 13 – Inicialização da base de dados



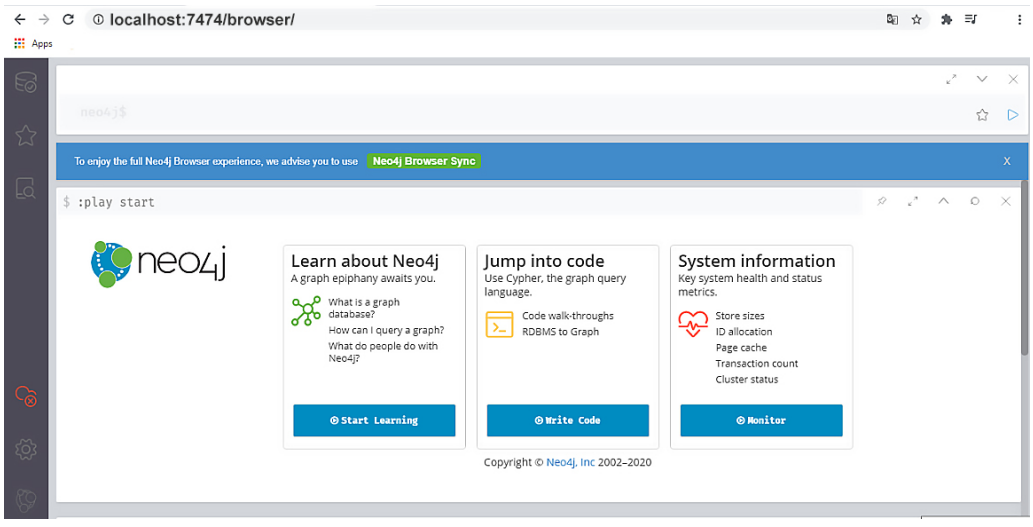

Vale destacar que o Neo4j permite a inicialização somente de uma base de dados por vez.

Após criada e inicializada a base de dados, no canto inferior esquerdo do Neo4j, é possível visualizar qual é a base de dados e qual projeto está ativo no momento. Para gerenciar essa base, devemos abrir o navegador web e informar o seguinte endereço:

- **Disponível em: <https://localhost:7474>.**

O endereço *localhost* corresponde ao acesso de uma base de dados local em nosso computador. A porta 7474 corresponde à porta padrão de acesso ao serviço do Neo4j. A Figura 14 apresenta a interface de gerenciamento web do Neo4j.

Figura 14 – Interface de gerenciamento web do Neo4j



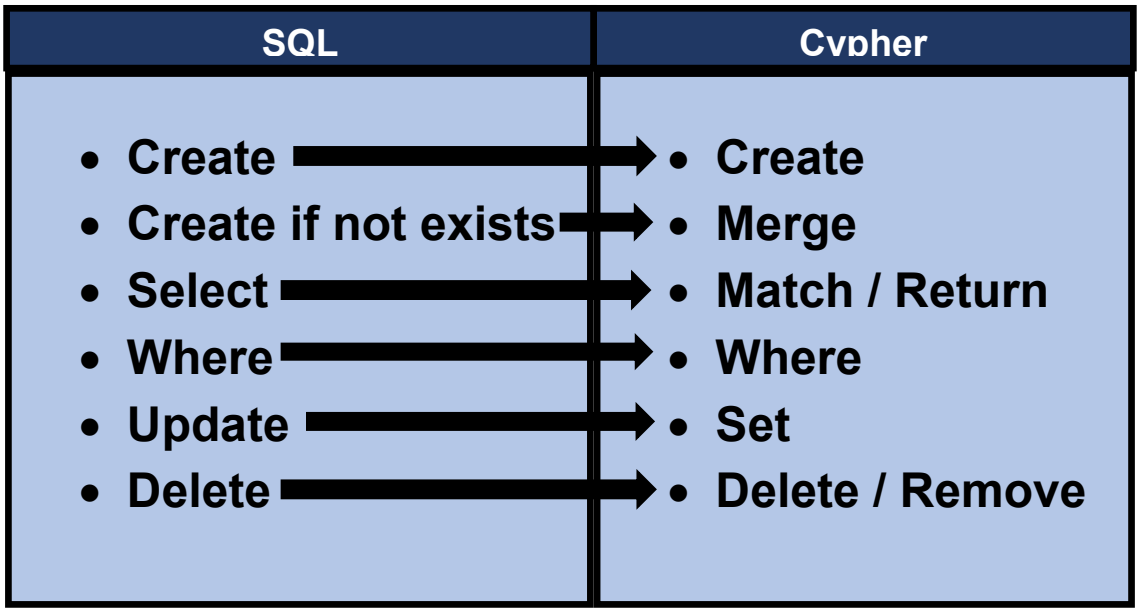
Ao acessar a interface de gerenciamento web, automaticamente ela estará conectada ao banco de dados inicializado no SGBD Neo4j Desktop. A partir desse momento, o gerenciamento do banco de dados, a criação de nós, relacionamentos, consultas, entre outros aspectos, serão executados pela interface web.

## TEMA 3 – GERENCIAMENTO DO BANCO DE DADOS NEO4J COM CYPHER

Para iniciar o tema, vamos retomar as nossas boas e velhas comparações entre bancos de dados NoSQL com o modelo relacional. Ao construir um banco de dados relacional, implementamos o modelo físico utilizando a linguagem SQL (Structured Query Language). O SQL está para os bancos de dados relacionais, assim como a linguagem Cypher está para os bancos de dados orientados a grafos. Conforme destacam Robinson, Webber e Eifrem (2013), inicialmente a linguagem Cypher foi desenvolvida para uso exclusivo do Neo4j, sendo posteriormente adotada por outros bancos de dados de grafo, através do projeto openCypher. É uma linguagem compacta e expressiva, projetada para ser de fácil leitura; seu uso é intuitivo, e os grafos geralmente são feitos em diagramas.

Conforme destaca Meyrelles (2015), o Cypher é a linguagem oficial de consultas do Neo4j. Ele permite criar, modificar e procurar dados em uma estrutura baseada em grafo de informações e relacionamentos. O Quadro 1 apresenta uma breve comparação entre os principais comandos da linguagem SQL e da linguagem Cypher.

Quadro 1 – Comparação entre as linguagens SQL e Cypher



Supondo uma consulta simples em um banco de dados relacional, para selecionar todos os registros de uma tabela, aplicaríamos uma consulta SQL do tipo:

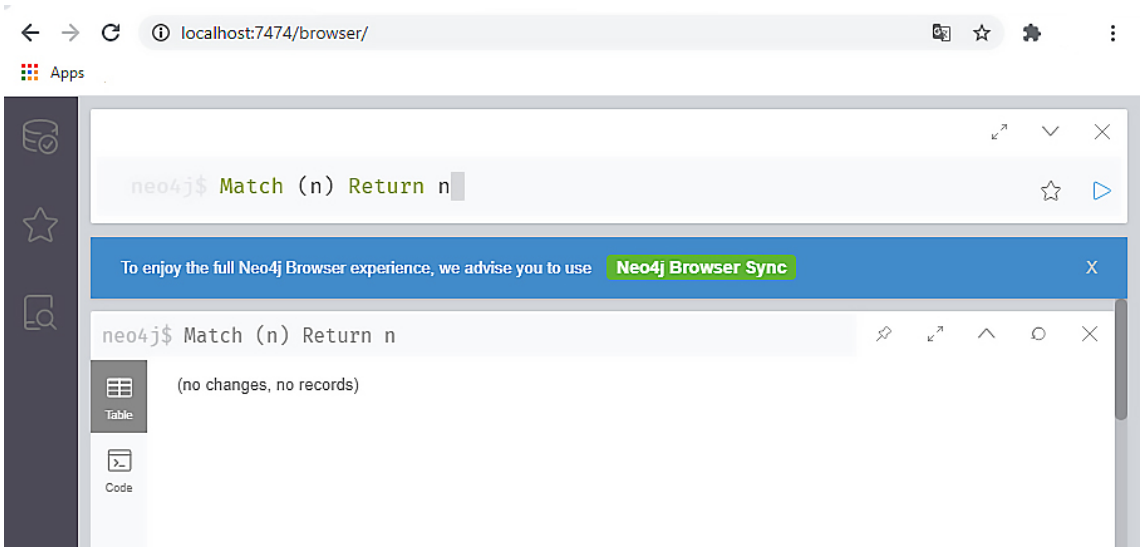
- **Select * From Nome_da_Tabela**

A mesma consulta em Cypher, para retornar todos os nós de uma base, seria escrita na forma:

- **Match (n) Return n**

Na consulta em Cypher, a letra “n” corresponde a uma variável que pode ser definida com qualquer nome. No exemplo, estão sendo selecionados todos os nós e armazenados em “n”, através do comando Match (n). Em seguida, esses nós são apresentados na tela através do comando Return n. A Figura 15 apresenta a execução da consulta de todos os nós em Cypher.

Figura 15 – Execução de consulta em Cypher




No exemplo da Figura 15, não é retornado nenhum nó, porque até esse momento nosso banco de dados chamado “Família” ainda está vazio.

## 3.1 Criando nós no banco de dados orientado a grafos

Vale lembrar que cada nó (ou *vértice*, como também é chamado) corresponde a um registro no banco de dados. Um nó pode ser adicionado isoladamente ou pode ser incluído em um “*Label*”, de modo que, para facilitar nossa compreensão, podemos associar cada *label* como sendo uma tabela em um banco de dados relacional.

1. **Criando um nó isoladamente**

    a. **Sintaxe:**

        i. Create (nome_do_nó)

    b. **Exemplo:**

        **i.** Create (Carlos)

2. **Criando nós em um Label**

    a. **Sintaxe:**

        i. Create (nome_do_nó :label)

    b. **Exemplo:**

        i. Create (Carlos :Pai)

        ii. Create (João :Pai)

        iii. Create (Maria :Mãe)

        iv. Create (Julia :Irmã)

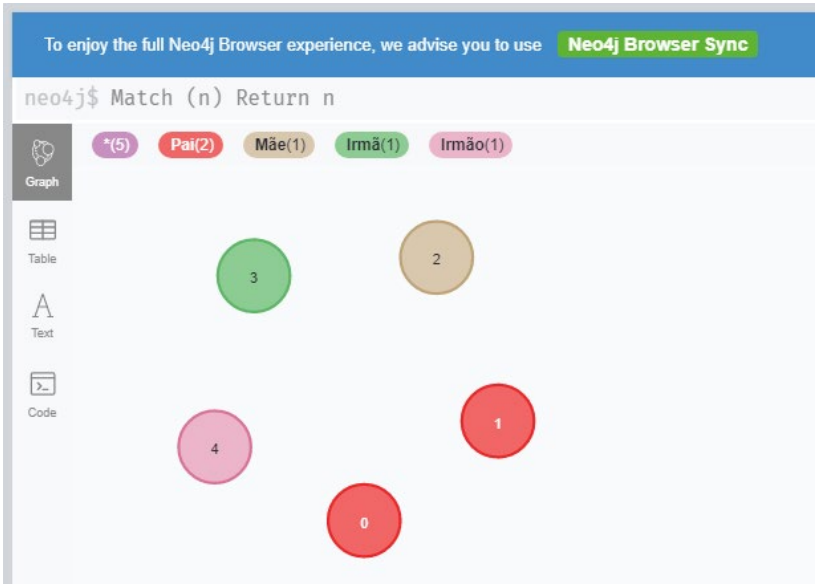
        v. Create (Mario :Irmão)

Um *label* também pode estar “dentro” de outro *label*, conforme o exemplo:

- Create (Carlos :Pessoa :Pai)

Conforme o exemplo, “Carlos” é um nó contido no *label* Pai, que por sua vez está contido no *label* Pessoa. A figura apresenta os nós do exemplo 2.b criados no Neo4j.

Figura 16 – Criação de nós no Neo4j



A numeração apresentada em cada nó corresponde ao "*id*" criado aleatoriamente de forma automática pelo Neo4j, porém esse valor pode ser alterado para qualquer uma das propriedades dos labels. Destaque também para os *labels* apresentados logo acima dos nós e para a associação de cores entre o *label* e os seus respectivos nós.

## 3.2 Adicionando propriedades aos labels

Para facilitar a nossa compreensão, uma propriedade, em um banco de dados orientado a grafos, corresponde a um atributo em uma tabela de um banco de dados relacional. Nesse contexto, um *label* pode ter vários atributos, como "idade", "telefone", "cpf", "altura" e "peso".

1. **Adicionando propriedades aos *labels***
    a. **Sintaxe:**
        i. Create (n :Label :Sublabel {propriedade:'Valor'}) Return n
    b. **Exemplo:**
        i. Create (n :Pessoa :Pai {nome:'Carlos', idade:'35'}) Return n
        ii. Create (t :Pessoa :Mãe {nome:'Maria', idade:'32'}) Return t

Vale frisar, nesse ponto, que ao contrário dos bancos de dados relacionais, em que cada atributo é único em uma tabela, nos bancos de dados orientados a grafos podemos repetir as propriedades em um mesmo *label*. Vamos analisar o caso de uma pessoa que tenha duas nacionalidades – por exemplo, brasileiro e espanhol. Nese caso, o nó poderia ser registrado da seguinte maneira:

- Create (n :Pessoa :Pai {nome:'Carlos', idade:'35', nacionalidade: 'Brasileiro e Espanho'}) Return n

O problema é que o registro "Brasileiro e Espanhol" é uma única *String*, não sendo, portanto, possível retornar o nó "Carlos" em uma simples consulta de todos os nós com nacionalidade brasileira. Assim, podemos representar o mesmo registro com duas ou mais propriedades iguais, conforme o exemplo:

- Create (n :Pessoa :Pai {nome:'Carlos', idade:'35', nacionalidade: 'Brasileiro', nacionalidade: 'Espanhol'}) Return n

## 3.3 Criando relacionamentos entre os nós

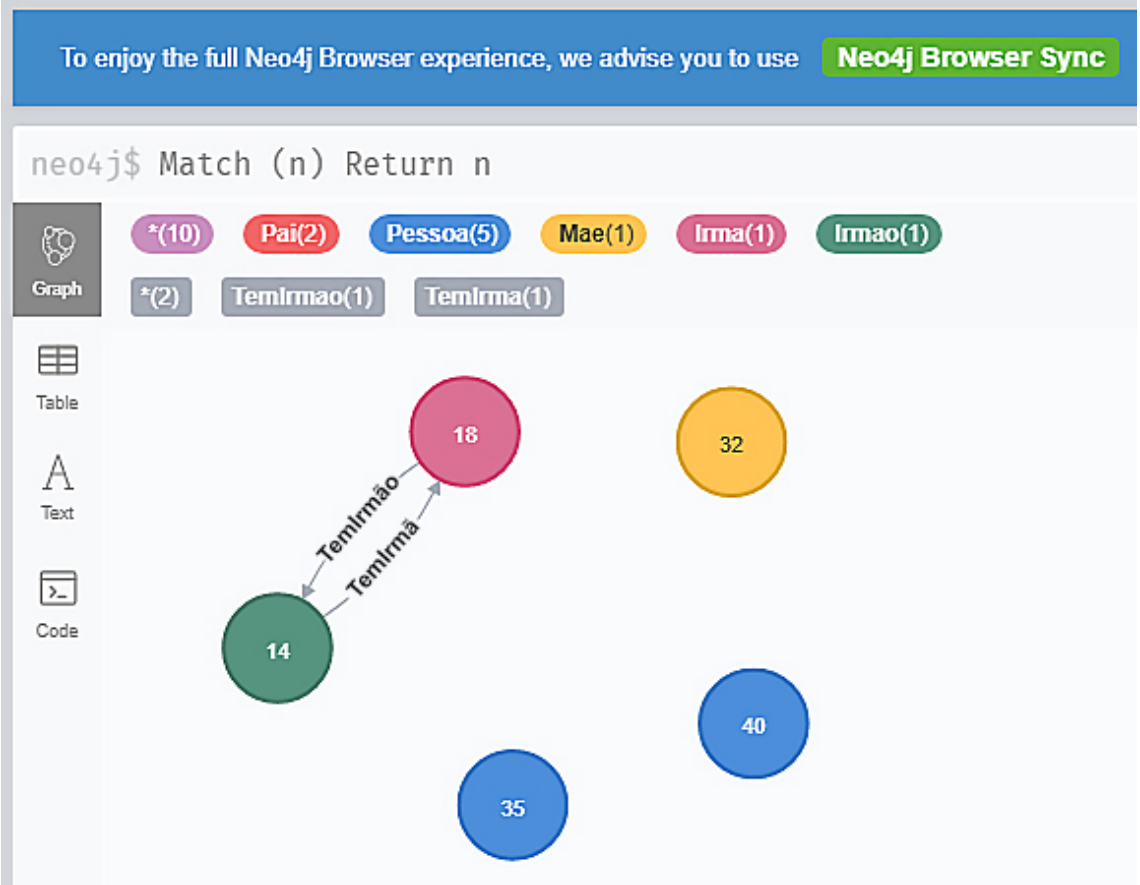
Novamente, podemos fazer uma comparação com o modelo de dados relacional, para nos ajudar a compreender melhor a estrutura de dados orientada a grafos. O relacionamento entre um nó e outro, em um banco de dados orientado a grafos, é similar ao relacionamento entre um registro de uma tabela com outro registro de outra tabela no modelo relacional, dado pela associação entre uma chave primária e uma chave estrangeira. A principal diferença reside no fato de que, no modelo orientado a grafos, inexiste o conceito de chaves primárias e estrangeiras, bastando que seja especificada a direção do relacionamento.

1. **Criando relacionamentos entre os nós**
    a. **Sintaxe:**
        i. Match (p :Label), (c :Label)
        Where p.propriedade = 'Valor' and c.propriedade = 'Valor'
        Create (p) - [r :Relacionamento] -> (c)
        Return p, c, r
    b. **Exemplo:**
        i. Match (p :Irmã), (c :Irmão)
        Where p.nome = 'Julia' and c.nome = 'Mario'
        Create (p) - [r :TemIrmão] -> (c)
        Return p, c, r
        ii. Match (p :Irmã), (c :Irmão)
        Where p.nome = 'Julia' and c.nome = 'Mario'
        Create (p) <- [r :TemIrmã] - (c)
        Return p, c, r

Como podemos ver no exemplo apresentado, os relacionamentos em um banco de dados orientado a grafos apresentam determinada direção. No exemplo “i”, criamos o relacionamento indicando que Julia tem Mario como irmão, porém não podemos afirmar o contrário nesse mesmo relacionamento, pois a direção aponta “->” de Julia para Mario. Já no segundo relacionamento do exemplo “ii”, realizamos o mesmo procedimento, porém agora o novo relacionamento aponta “<-“, de Mario para Julia; em contraste, alteramos o nome do relacionamento para se adequar à situação. A Figura 17 apresenta a estrutura

dos nós com os relacionamentos TemIrmão e TemIrmã entre os nós Julia e Mario.

Figura 17 – Representação dos relacionamentos em grafos




Vale frisar, nessa figura, que o Neo4j, ao invés de continuar representando cada nó com seu "id", agora está representado com o valor atribuído à propriedade "idade" de cada nó.

Se tentarmos criar um relacionamento já existente entre dois nós, utilizando a cláusula CREATE, esse relacionamento será duplicado no banco de dados. Para evitar isso, podemos utilizar a cláusula MERGE. A diferença entre essas duas cláusulas é que o Merge primeiro verifica a existência da estrutura que está sendo criada, para então criá-la caso ainda inexista no banco de dados.

**2. Criando relacionamentos com Merge**

a. Match (p :Irmã), (c :Irmão)
   Where p.nome = 'Julia' and c.nome = 'Mario'
   Merge (p) - [r :TemIrmão] -> (c)
   Return p, c, r

## 3.4 Criando consultas com filtro *Where*

Do mesmo modo como realizamos consultas com filtros específicos na linguagem SQL em um banco de dados relacional, utilizando a cláusula *Where*, podemos aplicar essa cláusula em Cypher.

1. **Criando consultas com filtro**
    a. **Sintaxe:**
        i. Match (n) Where n.propriedade = 'Valor' Return n
    b. **Exemplo:**
        i. Match (n) Where n.idade = '35' Return n
        ii. Match (n :Pai), (y :Mãe) Where n.idade = '35' and y.idade = '32' Return n, y

Voltando ao exemplo anterior, em que abordamos o caso de uma pessoa de duas ou mais nacionalidades, informações que seriam armazenadas como uma única *String* em uma propriedade, poderíamos sanar o problema de retornar todos os nós com nacionalidade brasileira utilizando a cláusula CONTAINS.

Dada a seguinte situação, já abordada anteriormente:

- Create (n :Pessoa :Pai {nome:'Carlos', idade:'35', nacionalidade: 'Brasileiro e Espanho'}) Return n

Poderíamos buscar todos os nós com nacionalidade brasileira utilizando a cláusula CONTAINS:

- Match (n) Where n.nacionalidade Contains 'Brasileiro' RETURN n

## 3.5 Apagando nós e relacionamentos

Para apagar do banco de dados os nós ou relacionamentos, usamos a cláusula DELETE em conjunto com a cláusula MATCH.

1. **Apagar todos os nós com relacionamentos**
    a. Match (n) Detach Delete n
2. **Apagar um relacionamento**
    a. Match (p :Irmã) - [r :TemIrmão] -> (c :Irmão)
    Where p.nome = 'Julia' and c.nome = 'Mario'
    Delete r
    Return p, c

## TEMA 4 – CARACTERÍSTICAS DE CONSISTÊNCIA, DISPONIBILIDADE E ESCALABILIDADE

Conforme aponta NEO4J (2014), entre as características do SGBD Neo4j, destaque para robustez, escalabilidade e alto desempenho. Há ainda a capacidade de garantir as propriedades ACID, que são uma das mais importantes características dos bancos de dados relacionais.

As propriedades ACID correspondem a quatro propriedades que um SGBD precisa para garantir a validade dos dados, mesmo que ocorram falhas durante o armazenamento ou problemas mais graves no sistema, como por exemplo problemas físicos em um servidor. Conforme destacado por Elmasri e Navathe (2005), essas propriedades são chamadas de propriedades ACID:

- **Atomicidade:** Todas as operações da transação são refletidas corretamente no banco de dados ou nenhuma delas;
- **Consistência:** A execução de uma transação isolada (sem qualquer outra transação) mantém a consistência do banco de dados;
- **Isolamento:** Embora várias transações possam ser executadas de maneira simultânea, duas transações $T_i$ e $T_j$ não sabem o que cada uma está executando;
- **Durabilidade:** Depois que uma transação é executada completamente com sucesso, as informações modificadas por ela persistem no banco de dados.

Parafraseando NEO4J (2014), a aplicação das propriedades ACID é a base para se garantir a consistência dos dados. Desse modo, todas as operações que modificam dados no Neo4j ocorrem dentro de uma transação, garantindo que os dados permaneçam consistentes.

De acordo com Neubauer (2010), um grafo de bilhões de nós e relacionamentos pode ser tratado em uma única instância de um servidor. Contudo, quando a capacidade de transferência/manipulação de dados é insuficiente, o banco de dados de grafo pode ser distribuído entre vários servidores, configurando um ambiente de alta disponibilidade.

## TEMA 5 – CASOS APROPRIADOS DE USO

Conforme observamos durante esta aula, uma das principais aplicações dos bancos de dados NoSQL orientados a grafos, dado o seu modelo estrutural, é em redes sociais.

> Bancos de grafos usam uma técnica chamada *Index free adjacency*, em que cada nó possui um endereço físico na memória RAM de nós adjacentes. Assim, os nós apontam para objetos aos quais estão relacionados. Isso representa um ganho significativo de velocidade nas consultas, já que não é preciso recorrer a um mecanismo central de indexação para extrair os relacionamentos entre os objetos. (Que tipo..., 2019)

Como vimos nos temas anteriores desta aula, os sistemas de banco de dados em grafos modelam seus dados por meio de nós (vértices) e relacionamentos (arestas), facilitando a modelagem de contextos complexos e definindo naturalmente relações existentes entre as entidades de uma base. Portanto, esses bancos de dados são melhor aplicados a situações em que os dados com os quais estamos trabalhando são altamente conectados, devendo assim ser representados a partir da firma como se conectam ou se correlacionam com outros dados.

Nessa abordagem, Penteado et al. (2014) apresentam como exemplos de uso dos bancos de dados orientados a grafos, além de redes sociais, sistemas que recomendam compras em lojas virtuais, sistemas que exploram dados químicos e biológicos para detecção de padrões, e sistemas *web,* como o PageRank da Google, que analisa a importância dos sites computando o número de arestas incidentes em cada site.

Outra aplicação a ser considerada para uso dos bancos de dados NoSQL, orientados a grafos, é encontrar padrões de conexão em dados que seriam difíceis de visualizar por meio de outras representações de dados, tais como sistemas para detecção de fraudes, que usam bancos de dados de grafos para analisar relações entre entidades que poderiam ser difíceis de encontrar de outras formas.

## FINALIZANDO

Nesta aula, abordamos de modo geral os conceitos e aplicações de uso sobre os bancos de dados NoSQL, orientados a grafos e suas principais características.

Conforme exposto por Robinson, Webber e Eifrem (2013), os bancos de dados NoSQL, orientados a grafos, são sistemas gerenciadores que, assim como os demais bancos de dados NoSQL, estudados nas aulas anteriores, fazem uso dos métodos de criação (*create*), leitura (*read*), atualização (*update*) e remoção (*delete*) – ou seja, baseiam-se nas propriedades CRUD, para manutenção e manipulação de objetos em bases de dados através de sistemas de controle transacional, de forma que a instância passe a ser trabalhada em memória.

Conforme o manual do Neo4j (2014), o SGBD que estudamos nesta aula, trata-se de um banco de dados orientado a grafo de código aberto, suportado comercialmente, cujo projeto foi elaborado com vistas a garantir confiabilidade e a otimizar a manipulação de estruturas em grafo, em vez de tabelas. Desse modo, uma aplicação que trabalha com Neo4j pode ter a expressividade de um grafo com a confiabilidade esperada de um banco de dados relacional.

Com base nessa abordagem, realizamos várias comparações entre a estrutura de dados orientada a grafos e o modelo de dados relacional, de modo a fixar as características de armazenamento e manipulação de dados, assim como os métodos utilizados para gerenciamento.

Na sequência, estudamos em profundidade o Neo4j, tido como um dos principais sistemas gerenciadores de bancos de dados NoSQL orientados a grafos. Abordamos sua instalação, características da interface do SGBD e a criação da nossa primeira base de dados. Depois, aprendemos a gerenciar essa base de dados com a linguagem Cypher.

Encerramos esta aula compreendendo as principais áreas de aplicação desses bancos orientados a grafos, e por que se adequam melhor a determinados contextos em comparação com outros modelos, como os relacionais.

## REFERÊNCIAS


ELMASRI, R.; NAVATHE, S. B. **Sistema de Banco de Dados**. São Paulo: Pearson Addison Wesley, 2005.

HECHT, R.; JABLONSKI, S. NoSQL Evaluation: A use case oriented survey. **International Conference on Cloud and Service Computing**, Hong Kong, p. 336-341, 2011.

MARQUESONE, R. **Big Data:** técnicas e tecnologias para extração de valor dos dados. São Paulo: Casa do Código, 2017.

MEYRELLES, M. Banco de dados orientados a grafos com Neo4j. **Accendis Tech**, 2015. Disponível em: <https://medium.com/accendis-tech/uma-gentil-introdu%C3%A7%C3%A3o-ao-uso-de-banco-de-dados-orientados-a-grafos-com-neo4j-ca148df2d352>. Acesso em: 1 maio 2021.

NEO4J. **The Neo4j Manual**. 2014. Disponível em: <http://neo4j.com/docs/stable/>. Acesso em: 1 maio 2021.

NEUBAUER, P. **Graph Databases, NOSQL and Neo4j**. 2010. Disponível em: <http://www.infoq.com/articles/graph-nosql-neo4j>. Acesso em: 1 maio 2021.

PENTEADO, R. M. et al. **Um Estudo sobre Bancos de Dados em Grafos Nativos**. São Francisco do Sul: Escola Regional de Banco de Dados ERBD, 2014.

QUE TIPO de Banco de Dados utilizar. **Solvimm**, 28 nov. 2019. Disponível em: <https://solvimm.com/blog/que-tipo-de-banco-de-dados-utilizar/>. Acesso em: 1 maio 2021.

ROBINSON, I.; WEBBER, J.; EIFREM, E. **Graph Databases**. Newston, MA: O'Reilly Media, 2013.