

Universidade Federal do Maranhão  
Engenharia da Computação  
Bancos de Dados II - Prof. Marcelo Vidigal  
Daniel Souza Dos Santos  
João Vitor Miranda Roma  
Mariana Andressa Luna Pinheiro

Atividade 6 - Banco de Dados Gráfico

São Luís - MA

2018

## Introdução

Os primeiros bancos de dados foram desenvolvidos na década de 60. A estrutura era rígida e era preciso conhecê-la para realizar consultas. O modelo relacional surgiu pouco depois, desconectando o modelo físico da estrutura lógica. E assim o modelo relacional se tornou padrão desde então.

Este modelo tem como característica obedecer todo o padrão ACID. Entretanto, não lida bem com grande quantidade de dados distribuídos. Dessa forma, um novo modelo foi proposto, para tratar com esse tipo de situação. Consiste no modelo de banco de dados NoSQL, abordado a seguir.

## Banco de dados NoSQL

Aplicações Web, suas demandas diferenciadas, redes sociais, entre outros aspectos do desenvolvimento da internet tem gerado um grande contingente de dados não estruturados. Para tratá-los e gerenciá-los, uma nova abordagem de banco de dados se tornou necessária, pois o modelo relacional tradicional não é adequado para tratar esse tipo de dado.

Desta forma, foi criado o modelo NoSQL (Not only SQL), que melhor atende às demandas de escalabilidade, alta disponibilidade e dados não-estruturados. Segundo o próprio criador do NoSQL, o modelo na verdade se trata de uma oposição ao modelo relacional, pois se propõe a tratar tudo o que este não lida bem.

Dentre as características principais dos bancos deste modelo, elucidadas a seguir, destacam-se: escalabilidade horizontal, ausência de esquema (ou esquema flexível), suporte nativo a replicação, API simples para acesso aos dados, consistência eventual.

- **Escalabilidade horizontal**

Trata-se de uma estratégia para lidar com o aumento dos dados. Consiste em expandir o poder de processamento do sistema, por meio do aumento de *threads* ou processos. Em geral é uma solução mais viável que a escalabilidade vertical (por *hardware*). Assim, com os modelos de bancos NoSQL, essa estratégia está disponível.

- **Ausência de esquema (ou esquema flexível)**

Como já mencionado, o modelo NoSQL não segue a estruturação de tabela presente nos modelos relacionais. Com esta característica, a escalabilidade é facilitada, bem como a disponibilidade. Porém não há tantas garantias de integridade, devido à estrutura mais flexível.

- **Suporte nativo a replicação**

Replicação é outra forma de prover escalabilidade. Consiste em copiar os dados em outros nós os dados adicionados. Com esta estratégia a leitura e a escrita são realizadas em diferentes nós, tornando a recuperação de dados mais rápida.

- **API simples para acesso aos dados**

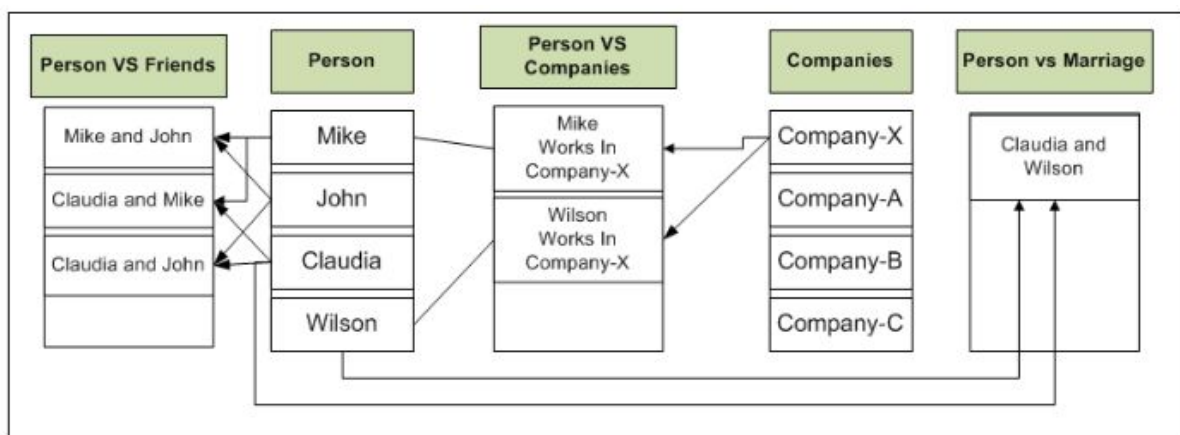
O modelo NoSQL não está tão interessado na forma como os dados são armazenados, e sim em sua disponibilidade. Assim, APIs para acesso de dados estão disponíveis de forma facilitada para recuperação dos mesmos.

- **Consistência Eventual**

Seguindo o teorema CAP, que diz que não é possível haver Consistência, Disponibilidade e tolerância a Partição ao mesmo tempo, no modelo NoSQL disponibilidade e tolerância a partição são privilegiados.

Bancos de dados relacionais têm sido uma das formas mais utilizadas e mais comuns de sistemas de software para o armazenamento de dados desde a década de 1970. Eles são altamente estruturados e armazenam dados na forma de tabelas, isto é, com linhas e colunas. Estruturar e armazenar dados na forma de linhas e colunas tem suas próprias vantagens; por exemplo, é mais fácil entender e localizar dados, reduzir a redundância de dados aplicando a normalização, manter a integridade dos dados e muito mais. Mas nem sempre esta é a melhor maneira de armazenar dados.

Vamos considerar um exemplo de rede social: Mike, John e Claudia são amigos. Claudia é casada com Wilson. Mike e Wilson trabalham para a mesma empresa. Na figura abaixo está uma das maneiras possíveis de estruturar esses dados em um banco de dados relacional:



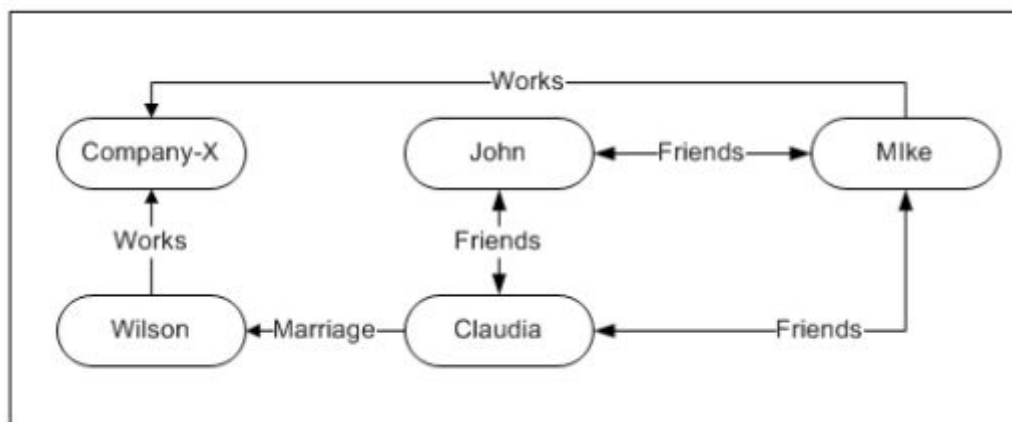
Bastante complexo, e pode ficar mais complexo ainda. Os relacionamentos evoluem e evoluirão ao longo de um período de tempo. Poderia haver novos relacionamentos, ou pode haver mudanças nos relacionamentos existentes. Pode-se projetar uma estrutura melhor, mas em qualquer caso, isso seria uma adaptação forçada do modelo em uma estrutura.

Bancos de dados relacionais (RDBMS) são bons para casos de uso em que o relacionamento entre entidades é mais ou menos estático e não muda ao longo de um período de tempo. Além disso, o foco do RDBMS é mais nas entidades e menos nas relações entre elas.

Pode haver muito mais exemplos em que o RDBMS pode não ser a escolha certa:

1. Modele e armazene 7 bilhões de objetos do tipo pessoa e 3 bilhões de objetos do tipo não-pessoas, para fornecer uma "visão de mundo" detalhada desde o planeta até uma calçada
2. Gerenciamento de rede
3. Genealogia
4. Ligações de transportes públicos e mapas de estradas

Considere outra maneira de modelar os mesmos dados:



Bem simples, este é um exemplo de um banco de dados gráfico.

Embora não exista uma definição única de grafo, a seguir está uma definição simples que nos ajuda a entender a teoria dos grafos:

“Uma estrutura de dados gráficos consiste em um conjunto finito (e possivelmente mutável) de nós ou vértices, junto com um conjunto de pares ordenados desses nós (ou, em alguns casos, um conjunto de pares não ordenados). Esses pares são conhecidos como arestas ou arestas. Como na matemática, diz-se que uma aresta  $(x, y)$  aponta ou vai de  $x$  para  $y$ . Os nós podem fazer parte da estrutura do grafo, ou podem ser entidades externas representadas por índices inteiros ou referências.”

O Neo4j, como um banco de dados gráfico de código aberto, faz parte da família NoSQL e fornece uma estrutura de dados flexível, em que o foco está nos relacionamentos entre as

entidades, e não nas próprias entidades. Sua primeira versão (1.0) foi lançada em fevereiro de 2010 e, desde então, nunca parou.

### **Introdução a bancos de dados gráficos**

Os bancos de dados relacionais não fornecem uma resposta boa o suficiente para os desafios reais de desenvolvimento de software e dados. No mundo de hoje, você precisa ser rápido; mais rápido que os outros. Isso significa que os dados são criados, atualizados e alterados mais rapidamente do que nunca. Os bancos de dados relacionais são inertes e não são projetados para lidar com mudanças rápidas e atender a novos requisitos de negócios, além de desenvolvimento ágil. Lidar com alterações em bancos de dados relacionais é muito caro e muito lento. Atingir escalabilidade e elasticidade também é um enorme desafio para bancos de dados relacionais. Atualmente, as mudanças ocorrem com frequência e a modelagem de dados é um grande desafio: você conhece apenas parcialmente os requisitos necessários e tem menos tempo para desenvolvimento e implantação do que nunca. Além disso, os bancos de dados relacionais não são adequados para processar dados altamente relacionados e aninhados ou objetos de aplicações hierárquicas.

Para lidar com enormes volumes de dados, bem como com uma variedade de dados e obter escalabilidade, nos últimos anos, as empresas começaram a usar soluções de banco de dados baseadas em NoSQL. Eles geralmente pertencem a uma das quatro categorias a seguir:

- Armazenamento de chave-valor: é o modelo NoSQL mais simples e flexível. Cada item no banco de dados é armazenado como uma chave, junto com seu valor. É muito eficiente, fácil de escalar e bom para alta disponibilidade. Exemplos são: Redis, Berkeley DB e Aerospike.
- Armazenamento de documentos: armazena uma estrutura de dados complexa para cada chave (documento). Os documentos podem conter vários pares de chave-valor diferentes, pares de matriz de chaves ou até mesmo documentos aninhados. Exemplos são: DocumentDB do Microsoft Azure, MongoDB, Elasticsearch e CouchDB
- Armazenamento de coluna ampla: armazena tabelas como seções de colunas de dados em vez de linhas. Pode ser descrito como um armazenamento de chave-valor bidimensional. Várias colunas são suportadas, mas você pode ter colunas diferentes em cada linha. Exemplos são: Cassandra e Hbase.

- Bancos de dados gráficos: são os sistemas NoSQL mais complexos e armazenam as entidades e as relações entre eles. Exemplos são: Neo4J, Azure Cosmos DB, OrientDB, FlockDB, DSE Graph e assim por diante.

Antes de começar com os bancos de dados gráficos, vamos fazer uma breve introdução à teoria dos grafos.

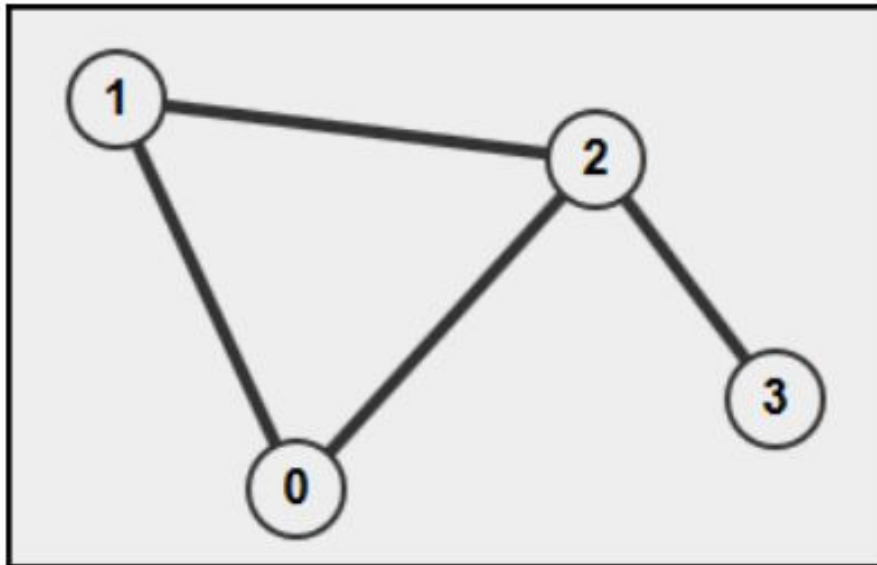
### O que é um grafo?

Os grafos são estudados na teoria dos grafos, uma parte da matemática discreta. Uma estrutura de dados do gráfico consiste em um conjunto finito de dois tipos de objetos:

- nós ou vértices
- bordas ou linhas, que são pares relacionados de nós

A definição formal e matemática de um grafo é exatamente isso:  $G = (V, E)$ . Isso significa que um grafo é um par ordenado de um conjunto finito de vértices e um conjunto finito de arestas.

Se todas as arestas dentro de um grafo forem bidirecionais, o grafo não será direcionado. Um grafo simples e não direcionado é mostrado da seguinte maneira:



Grafo não direcionado simples

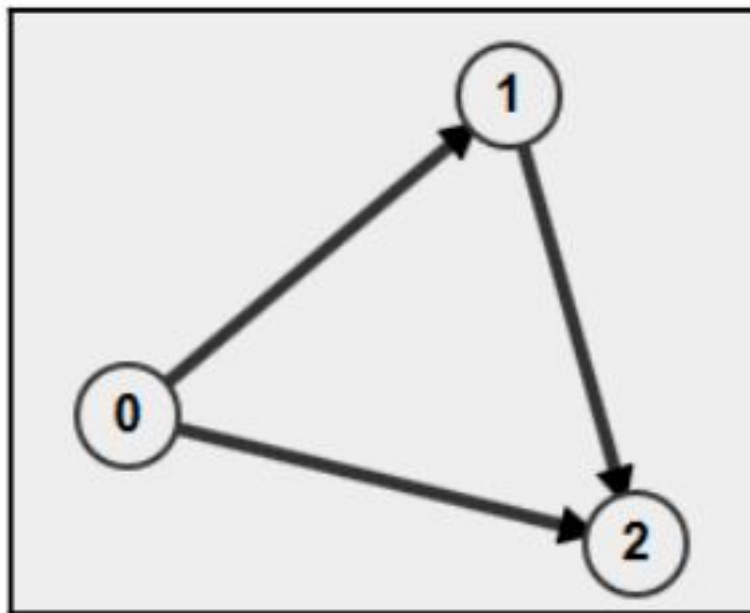
Este gráfico consiste em quatro nós e quatro arestas. O conjunto de nós e arestas pode ser representado com uma fórmula semelhante à seguinte:

$$V = \{0, 1, 2, 3\}$$

$$E = \{\{0, 1\}, \{0, 2\}, \{1, 2\}, \{2, 3\}\}$$

Chaves significam pares não ordenados (é possível viajar de um nó para outro em ambas as direções).  $\{0, 2\}$  é o mesmo que  $\{2, 0\}$ .

No caso de um grafo direcionado (dígrafo), as bordas podem ir apenas em uma direção:



Um grafo direcionado simples

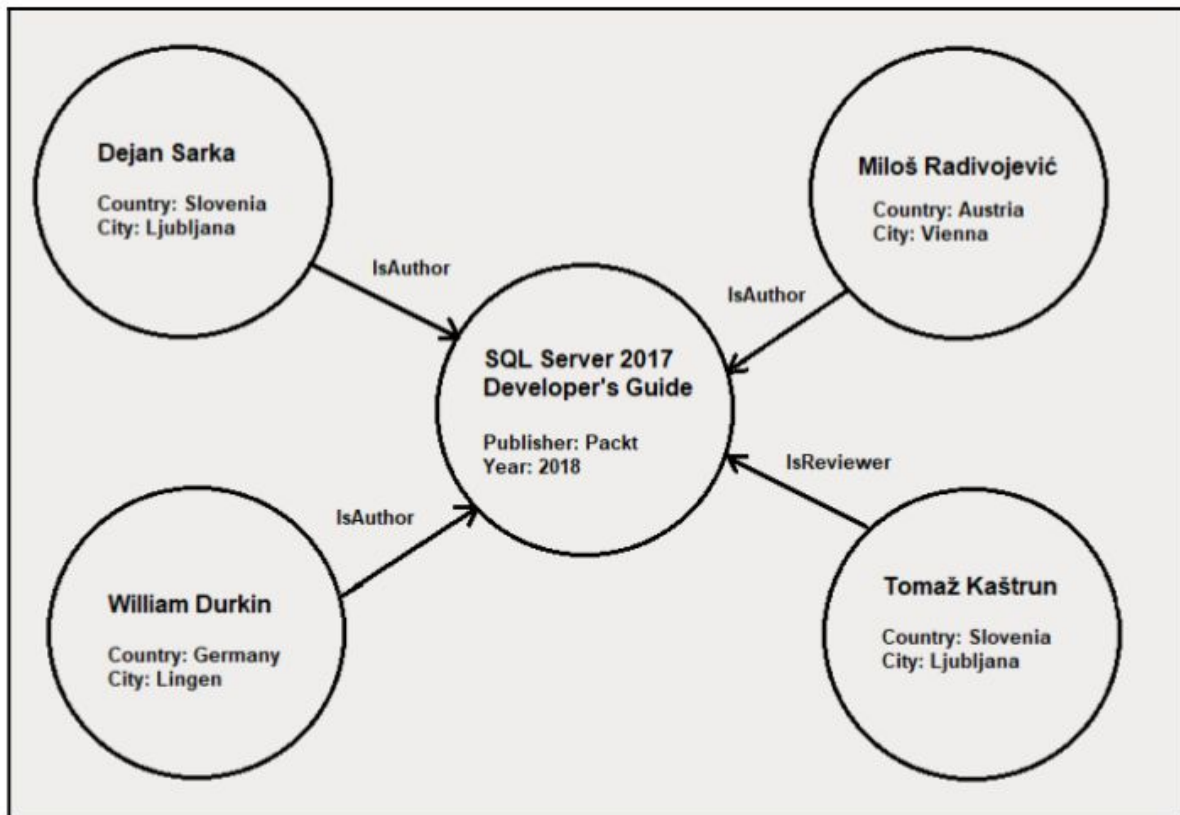
Este grafo consiste em três nós e três arestas, mas as arestas representam a relação em apenas uma direção. O conjunto de nós e arestas para esse grafo pode ser representado com uma fórmula semelhante à seguinte:

$$V = \{0, 1, 2\}$$

$$E = \{(0, 1), (0, 2), (1, 2)\}$$

Nós e arestas podem ter nomes e propriedades. Um modelo de gráfico que suporta propriedades é uma extensão dos grafos da teoria dos grafos, e isso é chamado de modelo de grafo de propriedades. Os nós e arestas podem ter nomes (rótulos para nós e tipos para relacionamentos), conforme mostrado na figura a seguir:



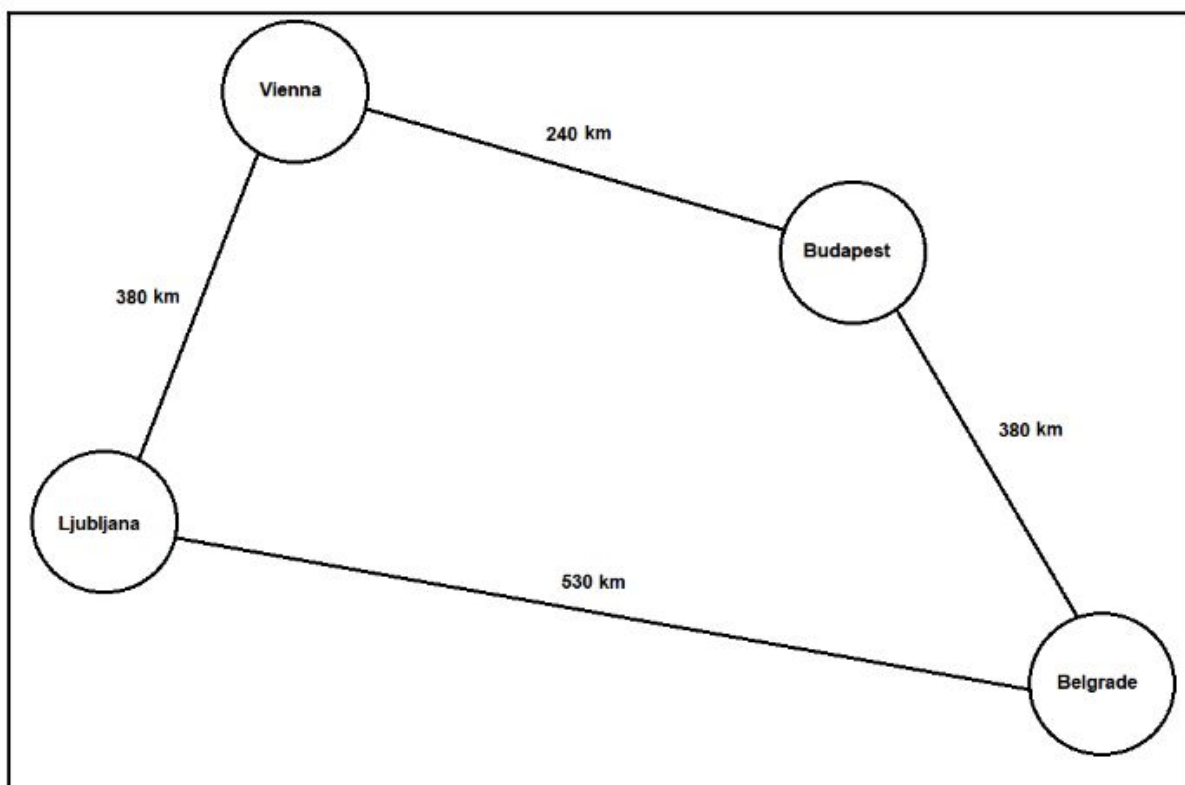


Um modelo de dados de grafo de propriedade simples

Este é um dígrafo, e você pode ver quatro nós representando quatro instâncias da entidade Pessoa e um nó representando a entidade Livro com seus nomes (ou rótulos) e pares de propriedades chave-valor adicionais. Existem dois tipos de arestas diferentes representando duas relações entre as pessoas e a entidade do livro. Todas as arestas são direcionadas.

Um modelo de grafo pode ser estendido atribuindo um peso às arestas. Esse grafo é chamado de grafo ponderado e geralmente representa estruturas nas quais as relações entre os nós têm valores numéricos, como o comprimento de uma rede rodoviária.

A figura a seguir mostra um grafo ponderado simples:



Um grafo ponderado simples

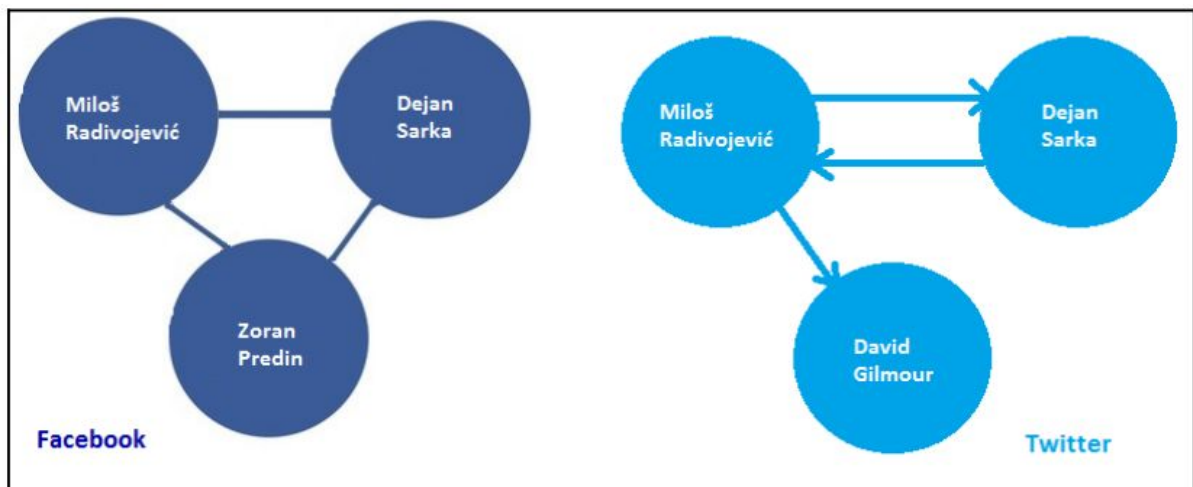
Como você pode ver, quatro nós representam cidades com a distância em quilômetros entre elas. Pode ser usado para encontrar o caminho mais curto entre duas cidades.

Os grafos são úteis para representar dados do mundo real. Existem muitas operações e análises úteis que podem ser aplicadas.

### **Teoria dos grafos no mundo real**

Os grafos são úteis para representar dados do mundo real. Muitos problemas práticos podem ser representados por gráficos. Portanto, eles estão se tornando cada vez mais significativos à medida que são aplicados a outras áreas da matemática, ciência e tecnologia.

Sua primeira associação com o uso da teoria dos grafos é provavelmente as redes sociais - conjuntos de pessoas ou grupos de pessoas com algum padrão de contato ou interações entre eles, como Facebook, Twitter, Instagram, LinkedIn e assim por diante. Aqui estão pequenos grafos que ilustram alguns usuários das duas primeiras redes sociais:



Grafos simples exemplificando relações em redes sociais

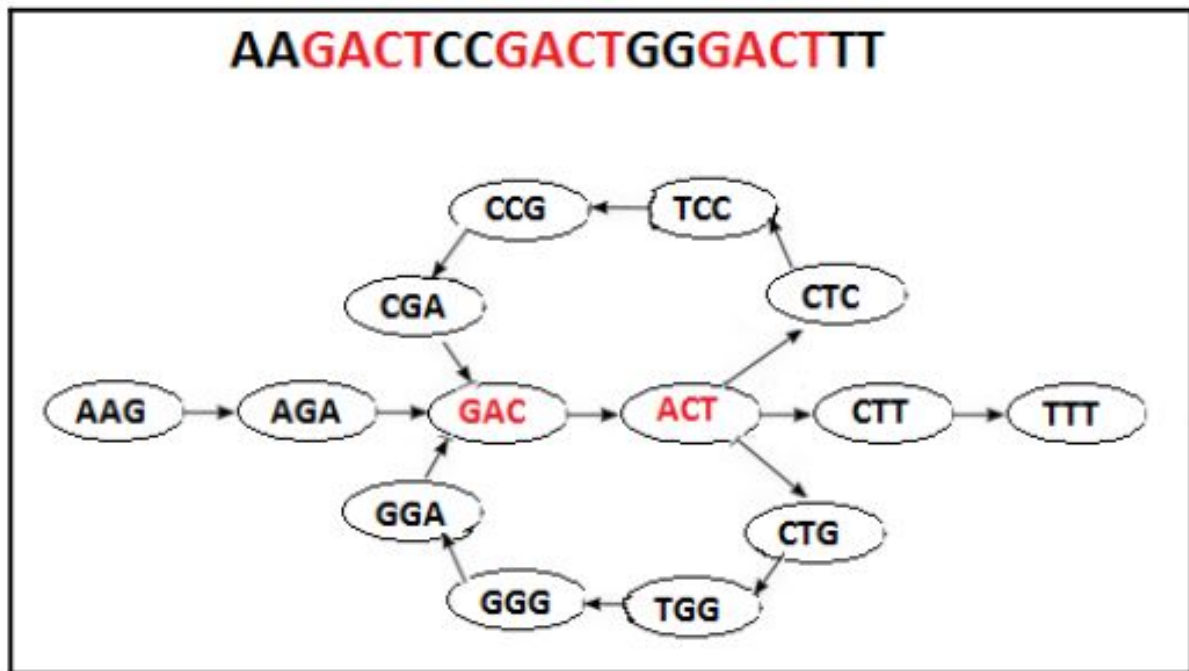
O grafo no lado esquerdo não é direcionado e representa os usuários conectados como amigos do ponto de vista do Facebook, enquanto o grafo que ilustra o Twitter é um dígrafo e mostra que Miloš Radivojević segue o perfil oficial de David Gilmour, mas infelizmente ele não segue ele de volta. A teoria dos grafos nas redes sociais é usada para implementar ferramentas como sugestões de amigos, amigos de amigos, para encontrar graus de conectividade, homofilia, redes de mundo pequeno, e analisar muitos aspectos das relações entre pessoas e pessoas e produtos ou serviços. Os resultados dessas análises são usados pelos mecanismos de recomendação. Ele também é usado para modelar muitas relações, como relações entre atores, diretores, produtores e filmes, por exemplo, no banco de dados do IMDb.

Em uma rede de computadores, a transmissão de dados é baseada no protocolo de roteamento, que seleciona as melhores rotas entre dois nós ou dispositivos. A teoria de grafos é usada em redes de computadores para encontrar o caminho melhor ou menos oneroso entre dois nós (algoritmo de caminho mais curto).

Nas redes de informação, a estrutura de links de um site pode ser representada por um grafo direcionado, com páginas da web como nós e links entre elas como arestas. Os rastreadores da Web executam algoritmos de pesquisa de grafo na Web e percorrem as redes de páginas da Web. O mecanismo de busca do Google usa um grafo no algoritmo PageRank para classificar as páginas resultantes.

Existem vários domínios biológicos e médicos em que as técnicas da teoria dos grafos são aplicadas: identificação de alvos de substâncias e determinação do papel de proteínas e genes de uma função desconhecida. Está sendo usado ativamente para a modelagem de redes

bio-moleculares, tais como redes de interação de proteínas, redes metabólicas, bem como redes de regulação transcricional. A teoria dos grafos é também um dos fatores mais importantes nas gerações de software de montagem de genomas. Um dos exemplos típicos de uso de grafos é o uso de grafos de Bruijn na montagem de genomas a partir do sequenciamento de DNA:



Montagem de genomas usando o grafo de Bruijn

A teoria dos grafos também é usada para estudar moléculas em química e física. Teoria Química dos Grafos (CGT, em inglês) é um ramo da química matemática que lida com aplicações não-triviais da teoria dos grafos para resolver problemas moleculares.

Encontrar o caminho mais curto entre dois pontos ou as melhores rotas possíveis nas redes rodoviárias é um dos problemas mais antigos em que os grafos são usados. A teoria dos grafos reduz as redes de transporte para uma coleção de nós (cidades e interseções rodoviárias) e arestas (estradas e ferrovias) com a distância entre dois nós como pesos para ajudar a encontrar a rota ideal ou a lista de pontos alcançáveis a partir do ponto inicial. Está sendo usada ativamente em sistemas de informações de companhias aéreas para otimizar as conexões de voos de um ponto de vista de distância e custo.

Existem muitas outras áreas onde a teoria dos grafos é aplicada: detecção de fraudes, mecanismos de recomendação, coletores de lixo em linguagens de programação (busca de dados inacessíveis), checagem de modelos (todos os estados possíveis e checagem do

código), checagem de conjunções matemáticas, quebra-cabeças e jogos, gerenciamento de identidade e acesso, Internet das Coisas (para modelar a conexão entre os componentes do sistema e os dispositivos IoT) e assim por diante.

### **O que é um banco de dados gráfico?**

Um banco de dados gráfico é um banco de dados que é construído sobre uma estrutura de dados de um grafo. Bancos de dados gráficos armazenam nós e arestas entre os nós. Cada nó e borda é identificado exclusivamente e pode conter propriedades (por exemplo, nome, país, idade e assim por diante). Uma borda também possui um rótulo que define o relacionamento entre dois nós.

Os bancos de dados gráficos exibem a mesma flexibilidade dos esquemas, o que é uma enorme vantagem, já que os esquemas atuais estão sujeitos a alterações: você pode adicionar novos nós sem afetar os existentes. Essa flexibilidade os torna apropriados para o desenvolvimento ágil. Eles são muito bons para recuperar dados altamente relacionados. Consultas em bancos de dados gráficos são intuitivas e semelhantes ao cérebro humano.

### **Quando você deve usar bancos de dados de gráficos?**

A maioria das coisas que os bancos de dados gráficos podem fazer também pode ser obtida usando um banco de dados relacional. No entanto, um banco de dados gráfico pode facilitar a expressão de certos tipos de consultas. Além disso, eles podem ter um melhor desempenho. Você deve considerar o uso de recursos de banco de dados gráficos quando seus aplicativos:

- Use dados hierárquicos
- Tem que gerenciar complexos relacionamentos muitos-para-muitos
- Analise dados e relacionamentos altamente relacionados
- Use dados aninhados intensivamente

Bancos de dados gráficos já existem há alguns anos. O padrão SQL: 2016 não define bancos de dados gráficos. No entanto, há vários anos, diferentes fornecedores começaram a fornecer soluções de banco de dados com recursos gráficos.

## **Mercado de bancos de dados de gráficos**

Atualmente, você pode encontrar vários bancos de dados de gráficos comerciais e de código aberto no mercado. Existem também algumas soluções específicas baseadas em grafos que não são verdadeiramente bancos de dados gráficos e são especializadas para lidar com problemas específicos de relacionamento, geralmente desenvolvidos por grandes empresas para resolver seus próprios problemas.

### **Neo4j**

O Neo4j é provavelmente o banco de dados gráfico mais popular. Ele é desenvolvido pela Neo4j, e é um banco de dados gráfico nativo: o Neo4j é construído do zero para ser um banco de dados gráfico. É o produto da empresa Neo Technology, com o Community Edition sob a licença GPL. Para clusters Neo4j altamente disponíveis e dimensionáveis, você precisa do Enterprise Edition. Neo4j é compatível com ACID. É baseado em Java, mas tem ligações para outras linguagens, incluindo Ruby e Python.

O Neo4j suporta sua própria linguagem de consulta Cypher, assim como o Gremlin. O Cypher suporta consultas analíticas de grafos avançados, como fecho transitivo, caminho mais curto e PageRank.

## Modelagem

Ambos os bancos foram modelados com uma estrutura de nós e arestas. Onde os nós foram subdivididos em dois grupos principais: pessoas e filmes, enquanto que as arestas das relações podem ser de quatro tipos:

- Pessoa dirigiu filme;
- Pessoa produziu filme;
- Pessoa escreveu filme;
- Pessoa atuou em filme.

A base de dados utilizada foi gerada aleatoriamente, tanto para os filmes e as pessoas envolvida quanto para as relações, sendo somente dez por cento das pessoas dirigiram, outros dez por cento produziram e outros dez por cento escreveram, sem que haja cruzamento entre essas três categorias, mas sem restrição para atuarem em um filme. Populados usando a plataforma Mockaroo (<https://mockaroo.com>) como geradora de dados aleatórios, com o devido cuidado para que os dados não se repetissem.

Para ambos os bancos foram criadas rotinas em python para gerar o Script SQL e CQL/Cypher para que estes pudessem ser usados para as medições dos seus tempos de execução nas plataformas SQL Server 2017 e Neo4j.

A base de dados ficou dividida em mil pessoas, cinco mil filmes e vinte e nove mil oitocentos e cinquenta e cinco relações, visto que a quantidade de atores em um filme variou entre um e quatro, representando o que seriam os atores principais.

### Modelagem do banco no Neo4j

Para a modelagem dos dados usou-se os padrões de inserção diretas. Para a inserção de pessoas e filmes, usou-se:

```
CREATE (p1:Pessoa {id:1, nome:"Nome e Sobrenome", nascEm:Ano})
CREATE (f1:Filme {id:1, titulo:"Título do filme", genero:"Genero do filme", lancamento:Ano, duracao:59})
```

Para as relações usou-se:

#### CREATE

```
(p111)-[:DIRIGIU]->(f1),(p662)-[:PRODUZIU]->(f1),(p513)-[:ESCREVEU]->(f1),  
(p772)-[:ATUOU_EM]->(f1),(p761)-[:ATUOU_EM]->(f1),(p251)-[:ATUOU_EM]->(f1),  
(p246)-[:ATUOU_EM]->(f1), (p637)-[:ATUOU_EM]->(f1)
```

### Modelagem no banco SQL Server

Seguindo a mesma lógica usada no Neo4j, apenas alterando o formato do código para gerar exatamente o mesmo banco, tem-se que primeiramente criar tabelas como nós (AS NODE) e como arestas (AS EDGE):

```
CREATE DATABASE Filmes;  
USE Filmes;  
CREATE TABLE Pessoa (  
    ID INTEGER PRIMARY KEY,  
    nome VARCHAR(100),  
    nascEm int  
) AS NODE;  
  
CREATE TABLE Filme(  
    ID INTEGER PRIMARY KEY,  
    titulo VARCHAR(100),  
    genero VARCHAR(100),  
    ano int,  
    duracao int  
) AS NODE;  
  
CREATE TABLE dirigiu AS EDGE;  
CREATE TABLE produziu AS EDGE;  
CREATE TABLE escreveu AS EDGE;  
CREATE TABLE atuouEm AS EDGE;
```

Para inserir os dados foram usados os comandos:

```
INSERT INTO Pessoa VALUES(id,'nome',nasc);  
INSERT INTO Filme VALUES(id, 'Título do Filme', 'Gênero', ano, duracao);  
INSERT INTO dirigiu VALUES(  
    (SELECT $node_id FROM Pessoa WHERE id =354),  
    (SELECT $node_id FROM filmes WHERE id = 1));
```



## COMPARAÇÃO DAS PERFORMANCES

Os testes foram realizados usando o mesmo computador com ambos os bancos instalados. As configurações do computador são:

<b>Marca</b>	Lenovo
<b>Linha</b>	IdeaPad
<b>Modelo</b>	320 80YF0007BR
<b>Processador</b>	Intel Core i5-7200U
<b>Velocidade do Processador</b>	2.5 GHz
<b>Memória RAM</b>	4096 MB DDR4 2133MHz
<b>Capacidade do HD</b>	500 GB
<b>Sistema Operacional</b>	Windows 10

Vale ressaltar que o número de processos sendo executados em segundo plano foi o menor possível. Bem como o shell do Neo4j foi preferido à interface gráfica.

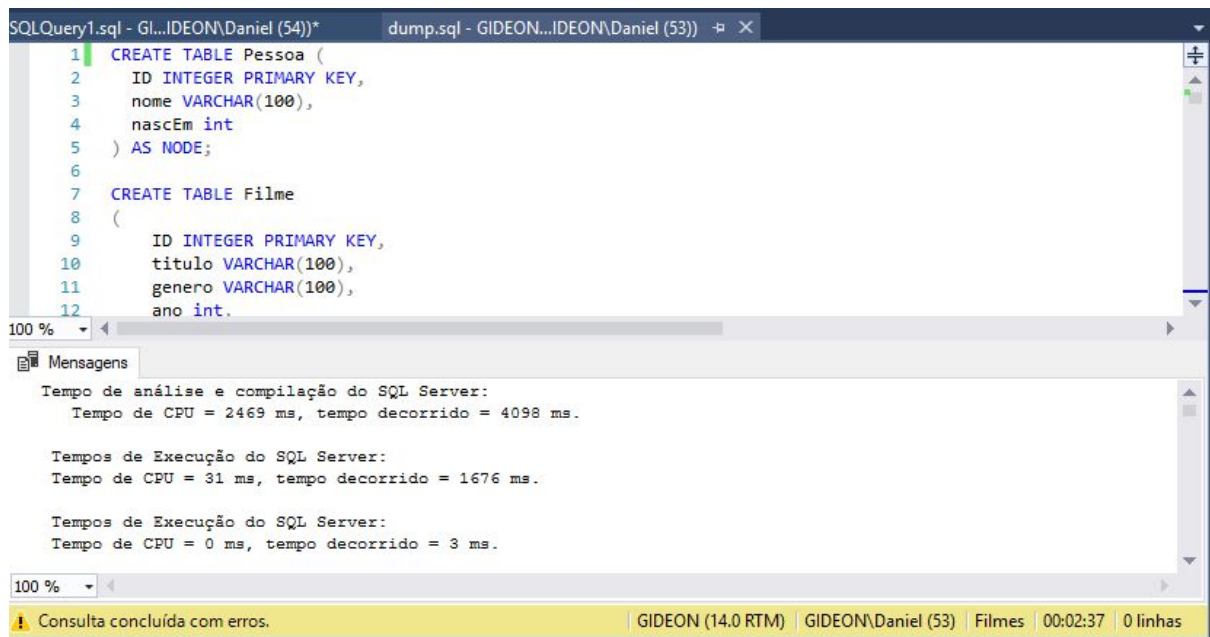
As primeiras análises foram realizadas em um banco menor e as comparações de performance em geral não foram suficientes para comparar os bancos, pois ambos apresentaram resultados próximos. Além de seus propósitos serem diferentes, assim como as suas arquiteturas de implementação.

Para contornar este problema e evidenciar a diferença, o banco teve um aumento significativo em número de nós e arestas, Mas sem que este número crescesse muito, pois o consumo de recursos do Neo4j impedia que algumas consultas pudessem ser feitas, pois causava estouro de pilha no sistema operacional.

Além disso, as consultas foram feitas primeiramente no SQL Server e convertidas para o Neo4j, já que este possui funções nativas para determinação de rotas entre dois pontos, assim como a menor rota ou até sobre iterar sobre as variáveis, características intrínsecas do Neo4j e ainda não implementadas no SQL Server.

### Inserção de dados

Os tempos de inserção nos bancos de dados foram bastantes distintos. Enquanto que o SQL Server levou apenas 157 segundos, o Neo4j levou mais de três vezes a mais este tempo gasto. 541 segundos, como ilustram as figuras abaixo:



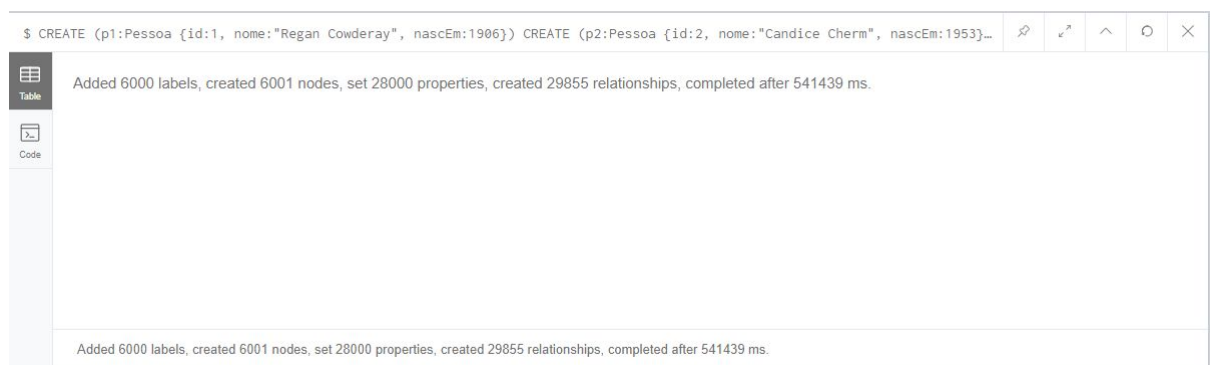
The screenshot shows the SQL Server Enterprise Manager interface. The top pane displays a SQL query window with the following code:

```
1 CREATE TABLE Pessoa (  
2     ID INTEGER PRIMARY KEY,  
3     nome VARCHAR(100),  
4     nascEm int  
5 ) AS NODE;  
6  
7 CREATE TABLE Filme  
8 (  
9     ID INTEGER PRIMARY KEY,  
10    titulo VARCHAR(100),  
11    genero VARCHAR(100),  
12    ano int.
```

The bottom pane shows the 'Mensagens' (Messages) window with the following output:

```
Tempo de análise e compilação do SQL Server:  
Tempo de CPU = 2469 ms, tempo decorrido = 4098 ms.  
  
Tempos de Execução do SQL Server:  
Tempo de CPU = 31 ms, tempo decorrido = 1676 ms.  
  
Tempos de Execução do SQL Server:  
Tempo de CPU = 0 ms, tempo decorrido = 3 ms.
```

At the bottom of the interface, a status bar indicates: 'Consulta concluída com erros. | GIDEON (14.0 RTM) | GIDEON\Daniel (53) | Filmes | 00:02:37 | 0 linhas'.



The screenshot shows the Neo4j Cypher Shell interface. The top bar displays the query: '\$ CREATE (p1:Pessoa {id:1, nome:"Regan Cowderay", nascEm:1906}) CREATE (p2:Pessoa {id:2, nome:"Candice Cherm", nascEm:1953})...'. The main area shows the execution result: 'Added 6000 labels, created 6001 nodes, set 28000 properties, created 29855 relationships, completed after 541439 ms.' The bottom status bar also displays: 'Added 6000 labels, created 6001 nodes, set 28000 properties, created 29855 relationships, completed after 541439 ms.'

## CONSULTAS SQL Server Versus NEO4J

Foram feitas quatro consultas diferentes para analisar o tempo que cada um dos bancos leva para que os resultados sejam mostrados na tela. Estas são:

1. Teste de tempo para listar quantos filmes cada diretor dirigiu.
2. Teste de tempo para listar em quantos filmes cada ator atuou.
3. Teste de tempo para listar com quem o ator que mais creditado já trabalhou.
4. Teste de tempo para listar quantos diretores foram dirigidos como atores pelo diretor que mais dirigiu filmes.

### 1. Teste de tempo para listar quantos filmes cada diretor dirigiu.

Este teste visa buscar todas as relações em que a pessoa dirigiu um filme e agrupar em um contador.

No SQL Server 2017:

```
SELECT p1.nome AS Diretor, count(Filme.titulo) AS [Número de Filmes]
FROM Pessoa p1, dirigiu d1, Filme
WHERE
    MATCH (p1-(d1)->Filme)
GROUP BY p1.nome ORDER BY [Número de Filmes] ;
```

Tempo de análise e compilação do SQL Server:  
Tempo de CPU = 16 ms, tempo decorrido = 19 ms.

(100 linhas afetadas)

Tempos de Execução do SQL Server:  
Tempo de CPU = 0 ms, tempo decorrido = 18 ms.  
Tempo de análise e compilação do SQL Server:  
Tempo de CPU = 0 ms, tempo decorrido = 0 ms.

No Neo4J, temos:

```
MATCH (n)-[:DIRIGIU]->(q) RETURN n.nome,count(*) ORDER BY count(*);
```

100 rows available after 36 ms, consumed after another 1 ms

Apesar de a diferença ser pequena, observa-se que o Neo4j apresentou os mesmos resultados em um período de tempo quase 30% menor.

## 2. Teste de tempo para listar em quantos filmes cada ator atuou.

De forma análoga ao primeiro teste, foram selecionados todos os atores e a quantidade de filmes em que estes participaram, para que se possa explorar mais as consultas, visto que uma maior quantidade de retornos pode evidenciar a diferença de performance.

No SQL Server 2017:

```
SELECT p1.nome as Ator, count(Filme.titulo) as [Número de Filmes]
from Pessoa p1, atuouEm d1, Filme
```

WHERE MATCH (p1-(d1)->Filme)  
group by p1.nome order by [Número de Filmes] ;

Tempo de análise e compilação do SQL Server:  
Tempo de CPU = 0 ms, tempo decorrido = 306 ms.

(1000 linhas afetadas)

Tempos de Execução do SQL Server:  
Tempo de CPU = 63 ms, tempo decorrido = 14841 ms.  
Tempo de análise e compilação do SQL Server:  
Tempo de CPU = 0 ms, tempo decorrido = 0 ms.

No Neo4j

MATCH (n)-[:ATUOU\_EM]->(q) RETURN n.nome,count(\*) ORDER BY count(\*);

1000 rows available after 359 ms, consumed after another 5 ms

Neste segundo teste, observa-se a indubitável superioridade do Neo4j em relação ao SQL Server, que precisou de menos de três por cento do tempo para exibir o mesmo resultado.

### 3. Teste de tempo para listar com quem o ator que mais creditado já trabalhou.

Após a realização do teste 2, foi escolhido o nó com o maior número de trabalhos como ator (Elisa Llewellyn, com 33 referências) sendo o nó inicial e listou-se todos os atores com quem este já atuou junto.

No SQL Server 2017:

SELECT p2.nome as Coator, Filme.titulo as [Filme]  
from Pessoa p1, atuouEm a1, Filme, atuouEm a2, Pessoa p2  
WHERE  
p1.nome='Elisa Llewellyn'  
and MATCH (p1-(a1)->Filme<-(a2)-p2)  
order by [Filme] ;

Tempo de análise e compilação do SQL Server:  
Tempo de CPU = 15 ms, tempo decorrido = 393 ms.

(124 linhas afetadas)

Tempos de Execução do SQL Server:  
Tempo de CPU = 0 ms, tempo decorrido = 573 ms.  
Tempo de análise e compilação do SQL Server:  
Tempo de CPU = 0 ms, tempo decorrido = 0 ms.

Neo4j

```
MATCH (n{name:"Elisa Llewellyn"})-[:ATUOU_EM]->(q)-[:ATUOU_EM]-(p) RETURN  
p.nome,q.titulo order by q.titulo;
```

91 rows available after 52 ms, consumed after another 0 ms

Observa-se que mais uma vez o Neo4j obteve um desempenho muito superior ao SQL Server. Houve uma diferença na quantidade de linhas exibidas nos resultados porque no SQL Server é necessário que haja uma cláusula explícita que o ator 1 seja diferente dele mesmo, o que acarretou em 124 resultados a mais (91 em participações e mais 33 próprios).

#### **4. Teste de tempo para listar quantos diretores foram dirigidos como atores pelo diretor que mais dirigiu filmes.**

O último teste foi realizado para que fosse percorrido um pouco do grafo e se resume ao diretor que mais teve créditos em filmes (Nola Haiden com 67 filmes creditados) que dirigiu filmes onde outros diretores participaram como atores.

No SQL Server:

```
SELECT distinct p2.nome as [Diretor], f1.titulo as [Título]  
from Pessoa p1, dirigiu d1, Filme f1, atuouEm a2, Pessoa p2, dirigiu d2, Filme f2  
WHERE  
    p1.nome='Nola Halden'  
    and MATCH (p1-(d1)->f1<-(a2)-p2)  
    and MATCH (p2-(d2)->f2);
```

Tempo de análise e compilação do SQL Server:  
Tempo de CPU = 31 ms, tempo decorrido = 138 ms.

(15 linhas afetadas)

Tempos de Execução do SQL Server:  
Tempo de CPU = 31 ms, tempo decorrido = 127 ms.  
Tempo de análise e compilação do SQL Server:  
Tempo de CPU = 0 ms, tempo decorrido = 0 ms.

No Neo4j:

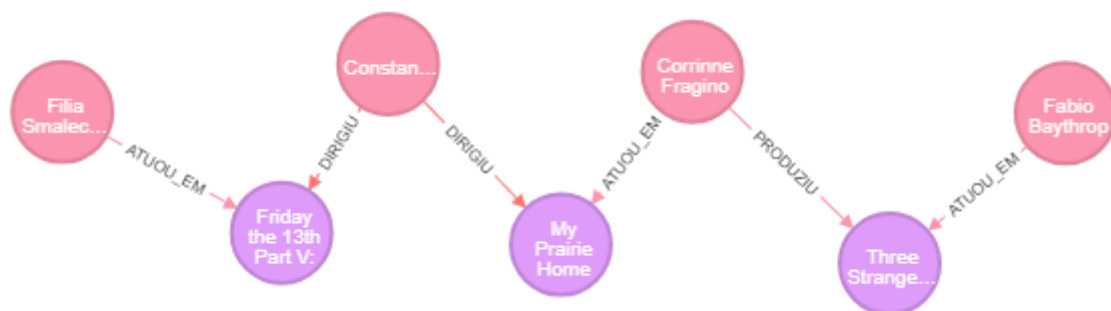
```
MATCH (n{name:"Nola Halden"})-[:DIRIGIU]->(q)<-[:ATUOU_EM]-(p)-[:DIRIGIU]->(v)
RETURN DISTINCT p.nome,q.titulo order by q.titulo;
```

15 rows available after 19 ms, consumed after another 0 ms

Neste último teste observou-se mais uma vez a superioridade do Neo4j. Depois de destes quatro testes, não se optou por mais testes diferentes, já que havia uma tendência do Neo4j ser superior nos seguintes. Mas como último teste, foi utilizado um recurso nativo do Neo4j, encontrar o menor caminho entre dois nós.

Para que o fosse exigido mais do Neo4j, foram usados os dois atores com menos créditos, “Fabio Baythrop” e “Filia Smalecombe”:

```
MATCH p=shortestPath((ator1:Pessoa {nome:'Fabio Baythrop'}) - [*] -
(ator2:Pessoa {nome:'Filia Smalecombe'})) RETURN p;
```



1 row available after 3 ms, consumed after another 13 ms

## CONSIDERAÇÕES FINAIS

Os cinco testes realizados mostram maior velocidade de execução no Neo4j, com exceção na inserção de dados no banco, mesmo a quantidade de caracteres e comandos serem mais curtos do que na linguagem SQL. Mostrando que, depois da inserção, o banco se mostra mais eficiente.

Os dados foram resumidos na tabela abaixo:

Item	SQL Server 2017	Neo4j
Inserção	157 segundos	541 segundos
Listagem de diretores	53 milissegundos	37 milissegundos
Listagem de atores	15210 milissegundos	364 milissegundos
Listagem de coatores	981 milissegundos	52 milissegundos
Listagem de diretor de diretores	327 milissegundos	19 milissegundos
Cálculo de menor caminho	---	13 milissegundos

Vale ressaltar que ajustes foram feitos para aumentar o tamanho do cache e da pilha do Neo4j para que este pudesse executar todos os comandos sem que fosse necessário eliminar o processo no sistema operacional. Além disso, outros ajustes mais profundos poderiam para que ambos bancos pudessem competir em pé de igualdade. Também é importante frisar que ambos bancos foram gerados com scripts não otimizados na inserção, o que pode ter sido crucial para o desempenho de ambos.

## **Referências Bibliográficas**

SARKA, Dejan; RADIVOJEVIC, Milos; DURKIN, William. SQL Server 2017 Developer's Guide: A professional guide to designing and developing enterprise database applications. 2018.

GUPTA, Sumit. **Building Web Applications with Python and Neo4j**. Packt Publishing Ltd, 2015.

Tutorials Point. **Neo4j Tutorial**. Disponível em: <https://www.tutorialspoint.com/neo4j/>. Acesso em 01/12/2018.